

AD-A182 031

THE AUTOMATED PROGRAMMING OF ELECTRONIC DISPLAYS(U)
SOFTWARE CONSULTING SPECIALIST INC FORT WAYNE IN
R W HASKER ET AL SEP 86 AFMIL-TR-86-3046

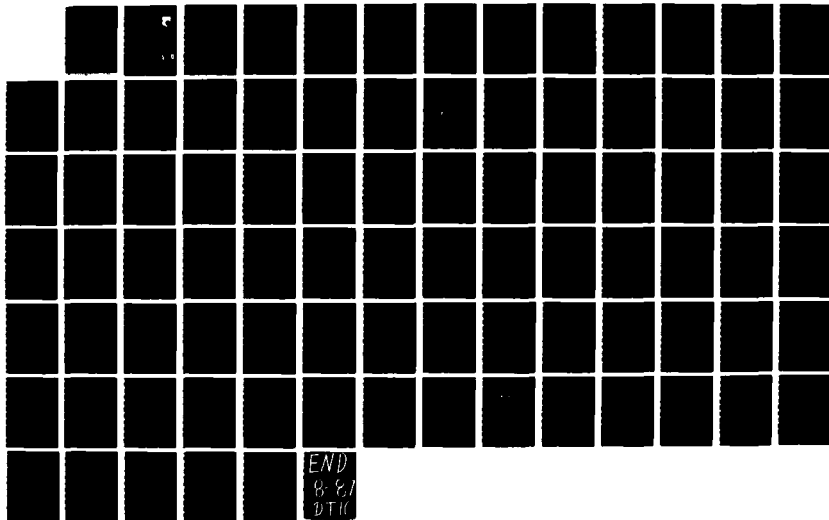
1/1

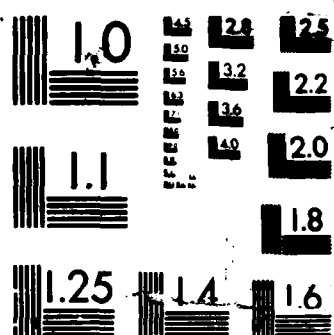
UNCLASSIFIED

F33615-85-C-3617

F/G 14/2

NL





MICROCOPY RESOLUTION TEST CHART

AFWAL-TR-86-3046

THE AUTOMATED PROGRAMMING OF ELECTRONIC DISPLAYS

R. W. Hasker
J. S. Edmondson
M. R. Fritsch

Software Consulting Specialists, Inc.
P. O. Box 15367
Fort Wayne, IN 46885

September 1986

Final Report for Period July 1985 - December 1985

Approved for public release; distribution is unlimited



AD-A182 031

FLIGHT DYNAMICS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6553

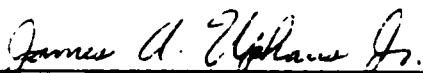
DTIC
ELECTE
JUL 01 1987
S E D


NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture use, or sell any patented invention that may in any way be related thereto.


This report has been reviewed by the Office of Public Affairs (ASD/PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.


JAMES A. UPHAUS, JR.
Electronics Engineer
Crew Systems Development Branch


RONALD I. MORISHIGE, Lt. Col, USAF
Chief, Crew System Development Branch
Flight Dynamics Division

FOR THE COMMANDER


RICHARD A. BOROWSKI, Lt Col, USAF
Chief, Flight Control Division

If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify AFWAL/FIGR, W-PAFB, OH 45433 to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFWAL-TR 86-3046	2. GOVT ACCESSION NO. ADA182031	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) THE AUTOMATED PROGRAMMING OF ELECTRONIC DISPLAYS		5. TYPE OF REPORT & PERIOD COVERED Final Report 1 July 1985 - 1 Dec 1985
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) R. W. Hasker J. S. Edmondson M. R. Fritsch		8. CONTRACT OR GRANT NUMBER(s) F33615-85-C-3617
9. PERFORMING ORGANIZATION NAME AND ADDRESS Software Consulting Specialists, Inc. P. O. Box 15367 Fort Wayne, IN 46885		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62201F 24030476
11. CONTROLLING OFFICE NAME AND ADDRESS Flight Dynamics Laboratory (AFWAL/FIGR) Air Force Wright Aeronautical Laboratories Air Force Systems Command, WPAFB, OH 45433		12. REPORT DATE September 1986
		13. NUMBER OF PAGES 87
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		16. DECLASSIFICATION/DOWNGRADING SCHEDULE
17. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution is unlimited		
18. DISTRIBUTION STATEMENT (of the abstract entered in Part 30, if different from Report)		
19. SUPPLEMENTARY NOTES		
20. KEY WORDS (Continue on reverse side if necessary and identify by block number) Video Display, Aircraft Display, Electronic Display, Real-time Display, Automated Programming, Graphics		
21. ABSTRACT (Continue on reverse side if necessary and identify by block number) ➤ A study was performed to determine the system requirements and top-level design for a system to support the design and automated programming of electronic displays for use in real-time environments. A hierarchy of levels was defined to support the creation and maintenance of display designs. Different attributes and capabilities were attached to each level of the hierarchy. The system was partitioned (continued)		

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

into an editor, animator, and code generator to support the creation, test, and compilation of display designs. The requirements for the major functions and data are discussed. A possible implementation is described which uses as much nondevelopmental hardware and software as possible to reduce the system's development time and total cost. Use of Ada to formally describe the display design and the use of computer-aided design (CAD) packages to edit display designs is discussed. The system feasibility is discussed and recommendations for system development are made.

Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

Table of Contents

<u>Section</u>	<u>Page</u>
1 Introduction	1
1.1 The Purpose	1
1.2 The Need	1
1.3 The Approach	3
2 System Description	5
2.1 A Brief Description	6
2.2 System Partitioning	6
2.3 Data Requirements	9
2.3.1. Display Design Requirements	9
2.3.1.1 The Display Design Hierarchy	9
2.3.1.2 Entities	14
2.3.1.2.1 Attributes	15
2.3.1.3 Segments	16
2.3.1.3.1 Dynamic Controls	18
2.3.1.4 Modules	24
2.3.2 Formal Display Description Requirements	25
2.3.3 Executable Display Description Requirements	26
2.3.4. Library Requirements.	26
2.3.5 Test Data Requirements	27
2.3.6 Target Specific Code Requirements	27
2.4 Process Requirements	28
2.4.1 Editor Requirements	28
2.4.1.1 General Editing Capabilities	28
2.4.1.2 Default Values	30
2.4.1.2.1 Default Attribute Values	30
2.4.1.2.2 Default Dynamic Control Values	31
2.4.2 Animator Requirements	31
2.4.3 Code Generator Requirements	32
2.5 Hardware Requirements	33
2.5.1 Input Device Requirements	34
2.5.2 Graphics Terminal Requirements	35
2.5.3 Hard-copy Printer Requirements	35
2.5.4 Host Computer Requirements	35
2.5.5 Target Display Device Requirements	35
2.6 System Usage Requirements	36
2.6.1 Overview	36
2.6.2 General Usage Requirements	36
2.6.3 Using the Editor	37
2.6.4 Using the Code Generator	37
2.7 System Expansibility Requirements	38
3 System Implementation	39
3.1 The Problems of Implementation	39
3.1.1 Current Technology and Its Limitations	40
3.1.2 Circumventing These Limitations	42

Table of Contents (continues)

3.1.2.1	Use of Hardware	42
3.1.2.2	Reduction of Requirements	43
3.1.3	System Implementation Partitioning	44
3.2	Data Implementation	44
3.2.1	Display Design Implementation	44
3.2.2	Formal Display Description Implementation	45
3.2.3	Executable Display Description Implementation	47
3.2.4	Library Implementation	47
3.2.5	Test Data Implementation	47
3.2.6	Target-Specific Code Implementation	48
3.3	Process Implementation	48
3.3.1	Editor Implementation	49
3.3.1.1	The Graphics Editor	49
3.3.1.2	The Dynamics Editor	52
3.3.1.3	The Translator	53
3.3.2	Animator Implementation	53
3.3.2.1	The Animation Compiler	54
3.3.2.2	The Animation Processor	54
3.3.3	Code Generator Implementation	55
3.3.3.1	The Target-Specific Compiler	55
3.3.3.2	The Target-Specific Linker	56
3.4	Hardware Implementation	56
3.4.1	Input Device Implementation	57
3.4.2	Output Device Implementation	58
3.4.2.1	Hard-copy Output Device Implementation	59
3.4.2.2	Graphics Terminal Implementation	59
3.4.3	Host Computer Implementation	60
3.5	A Brief Description of System Implementation	61
4	Conclusions and Recommendations	62
APPENDIX A GLOSSARY OF TECHNICAL TERMS		63
APPENDIX B DETAILED OPERATIONAL SCENARIO: CREATING AN EXAMPLE DISPLAY		68
B.1	Introduction	68
B.2	Sketching the Display	68
B.3	Determining the Display Parameters	68
B.4	Partitioning the Design	69
B.5	The Gauge Module	69
B.5.1	Segment 1.0	70
B.5.2	Segment 1.1	71
B.5.3	Segment 1.2	71
B.5.4	Segment 1.3	72
B.6	The Top Level Module	72
B.6.1	Segment 1.0	72
B.6.2	Segment 1.1	72

Table of Contents (concluded)

B.6.3	Segment 1.2	73
B.6.4	Segment 1.3	73
B.7	Conclusion	73
APPENDIX C COMPUTER-AIDED DESIGN PACKAGE AND HARDWARE SURVEY RESULTS.		74
C.1	Introduction	74
C.1.1	The Hardware	74
C.1.2	The Software	74
C.2	The Survey	75
C.2.1	Graphics Terminals	76
C.2.2	Host Computers	76
C.2.3	Other CAD Package	76
C.3	Conclusion	76
APPENDIX D AN ASSESSMENT OF GRADS		77
D.1	Introduction	77
D.2	A Description of GRADS	77
D.3	The Advantages of GRADS	78
D.3.1	Similar Alternatives	78
D.4	The Disadvantages of GRADS	79
D.5	Conclusion	79

Table of Figures

Figure	Title	Page
1-1	The Display Environment	2
1-2	The Current Process for Creating Displays	3
2-1	The Proposed Process for Creating Displays	5
2-2	The System Partitioning	7
2-3	The Display Design Hierarchy	10
2-4	Characteristics Associated with Hierarchy Levels	10
2-5	A Simple Display	12
2-6	The Structure of the Simple Display Using the Hierarchy	13
2-7	The Intersection of Clipping Boundaries	17
2-8	The Segment Numbering System	18
2-9	The Priorities of Segments	19
2-10	The Cumulative and Noncommutative Nature of Transformations	21
2-11	The Use of User-specified Priorities	22
3-1	The Editor Subsystem Partitioning	49
3-2	The Animator Subsystem Partitioning	54
3-3	The Code Generator Subsystem Partitioning	56
B-1	The Fluid Status Display	69
B-2	Segment 1.0	70

1 Introduction

1.1 The Purpose

This report documents a study made to determine the requirements for and the feasibility of a system designed to automate the programming of electronic graphic displays which respond to real-time inputs. It is directed at those responsible for designing such displays. This study was performed by Software Consulting Specialists, Inc., Fort Wayne, Indiana, for the Air Force Wright Aeronautical Laboratories, Flight Dynamics Laboratory at Wright-Patterson Air Force Base under contract number F33615-85-C-3617 and project number 24030476.

1.2 The Need

Complex graphics displays that gather and display rapidly changing information are seeing increased use in time-critical environments. The ability of these displays to efficiently present large amounts of information makes them ideal in situations that require an operator to quickly assimilate and act on real-time data. Information that would otherwise have to be presented in numerical form or on dials and gauges can, via a multi-purpose display, be shown as a graph, drawn on a diagram, overlaid on a map, or displayed in a three-dimensional scene.

It is the responsibility of a graphics designer to select the best method of presenting this information on the display. The display design process is made difficult, however, by the large amount of time currently needed for the designer to convert a display concept into the final product--with the assistance of a programmer. The designer gets no immediate feedback about the display appearance or operation. Since the display will move and change in response to real-time inputs, the final operation of the display design is often never seen until it is implemented in the target display device, perhaps months later. The evaluation and improvement of display designs at this late stage is inefficient and costly. Thus, many designs are never optimized because of the time and cost involved. What is needed is a system that allows a display designer to interactively construct and modify display designs, test their operation using simulated real-time inputs, and then translate this design into the necessary code(*) to run the target display device.

* Underlined words are defined in the Glossary of Technical Terms, Appendix A.

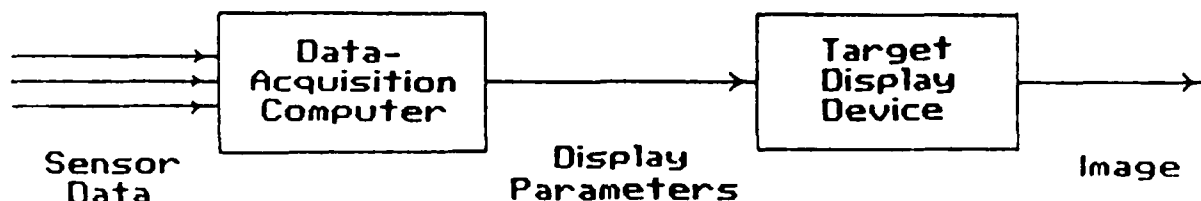


Figure 1-1 The Display Environment

The environment for a typical target display device (as shown in Figure 1-1) is as follows: a data-acquisition computer gathers data from sensors and other subsystems and transmits the display parameters to a graphics device. The graphics device then draws an image on the screen which reflects the values of these parameters. The graphic image can present the information in a variety of formats, including representations of instrument panels, symbolic schematics showing the state of the environment, and views of the outside world in both two and three dimensions.

The current process of creating display designs is illustrated in Figure 1-2. It is very time consuming, involving both a designer and a programmer working in series. The designer first draws a rough sketch of each display to be created. The sketch is refined into a drawing which is given to a computer programmer along with the designer's explanation of how the display should operate. The programmer writes a computer program for the target device's microprocessor which will make the electronic screen display the drawing. This computer program is transferred to the target display system hardware in the target environment or an appropriate simulator. The designer must then evaluate the actual display in this environment. Further refinements are given to the programmer who then does the necessary reprogramming. This process is repeated until the display is acceptable to the designer or until available time and funds are exhausted.

A primary problem with this display design process is the lack of communication between the designer and the programmer: the designer's sketch is too informal to ensure that the programmer's product will be correct. As a consequence of the inexactness of the drawing, some components in the final display will likely be shaped or placed incorrectly. Also, it is very difficult to show the movements of the various display components on the sketch. When the programmer does not fully understand the designer, he will often make a reasonable but not necessarily correct guess at what is desired. Since the programmer is rarely an expert in designing optimal displays, he often needs advice from designers and other programmers.

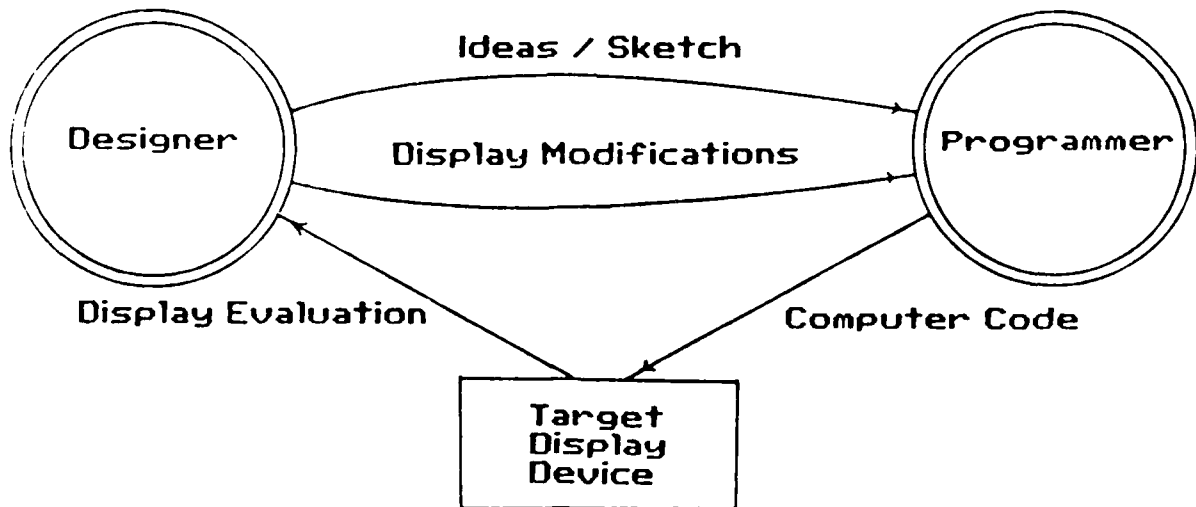


Figure 1-2 The Current Process for Creating Displays

If the communication problem were solved by improving the interface between the designer and the programmer, other problems would still remain. The designer must wait for the programmer to complete the implementation before he can begin to evaluate the design. It must be possible to rapidly iterate between designing and evaluating. This would allow the designer to maximize his creativity by minimizing the interruptions in his flow of thought. Also, programmers must repetitively re-solve the same electronic and programming problems every time a new display device technology is used. These side issues divert resources from the main design effort.

Anyone confronted with creating displays for real-time environments must deal with the above problems. Large amounts of time and money are spent each year, in both the government and private sectors, producing graphic display designs. A system which could substantially improve the efficiency of producing these displays would result in substantial savings for its users. Software Consulting Specialists, Inc. has performed this study to determine the requirements for and the feasibility of a system designed to support the design and automated programming of these electronic graphic displays. Whenever it is important to minimize time and funds required to create new graphic displays, this system will be a valuable tool to help display designers accomplish their goal.

1.3 The Approach

The requirements for this system and a top level design were completed using a top-down approach. The system described in this report is not configured for any particular hardware or software. It is rather an ideal, implementation-independent system defined to meet the needs. The detailed

system design must wait until the next phase of this effort, at which time all nondevelopmental hardware and software will be acquired and any gaps filled with custom hardware and software. The actual implementation of the system may result in the temporary exclusion of several system requirements due to current technological limitations. These can be added as the development of new technology permits.

We performed the following system specification steps:

- o Define the generic system requirements, which include functional, performance, and operational requirements;
- o Define the operational use of the system;
- o Prepare a top level design of the system;
- o Briefly determine the availability and suitability of existing hardware and software components; and
- o Determine where custom components will probably be needed to produce a complete system.

In accomplishing the above, we kept in mind several key guidelines which apply to the development of any major system. Foremost is that the system requirements are defined by the user and the system designer, rather than by the capabilities of current technology. The system should be minimal; no capabilities should exist which do not fit the need. The system must also be modular to minimize maintenance costs. The system must be open ended to allow the addition of new requirements with a minimum impact on the design. Finally, as much commercially available equipment as possible must be used to reduce the development time and cost.

We present our findings in the following sections and appendices. Section 2 specifies the requirements for an ideal system based on the need. Section 3 explains which features of the ideal system are unsupported by the current technology and presents a possible implementation based on the reduced requirements. Section 4 presents our conclusions. Appendix A is a glossary of many of the technical terms used in the report. Appendix B presents a sample design session. Appendix C presents the results of a hardware and software survey of possible nondevelopmental items. And Appendix D assesses the use of GRADS (Graphics Real-Time Application Display Support) in the implemented system.

2 System Description

A new process for creating displays is proposed to fill the above needs. This process is illustrated in Figure 2-1. The designer first creates a formal description of his display design. The designer then tests the display design by putting it into motion. Finally, the executable computer program for a particular target display device is created by a code generator. This section of the report explains the process in detail.

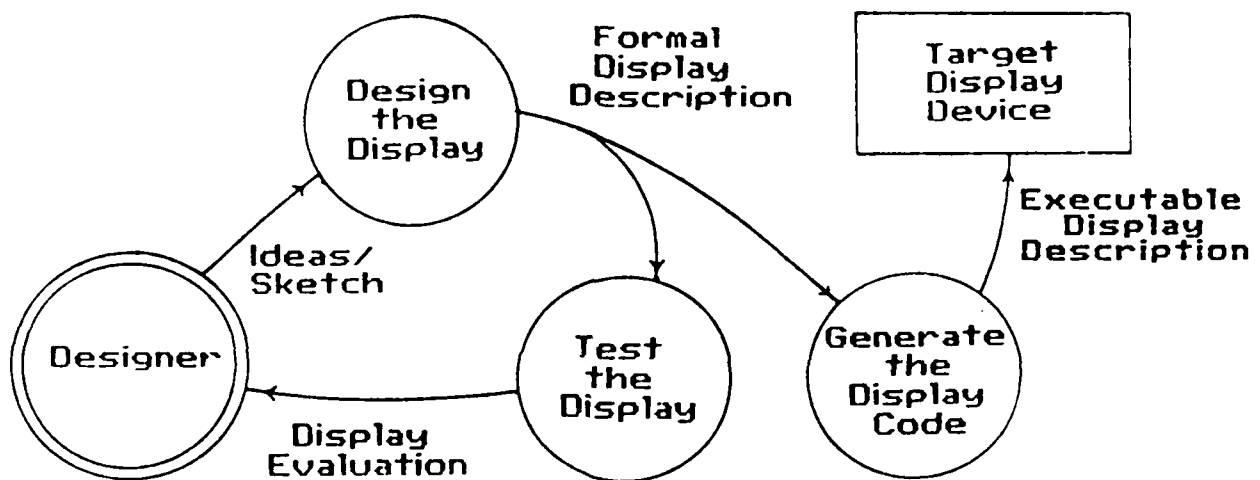


Figure 2-1 The Proposed Process for Creating Displays

This section is divided into the following seven subsections:

- o A Brief Description,
- o System Partitioning,
- o Data Requirements,
- o Process Requirements,
- o Hardware Requirements,
- o System Usage Requirements, and
- o System Expansibility Requirements.

The brief description overviews the entire system, breaking it into discrete components and describing how they interface with each other. The next four sections cover in detail the requirements for the components and interfaces mentioned in the brief description. The next section explains how the system will be used as a whole and as the various parts. The last section specifies the areas in which the system should be expansible.

2.1 A Brief Description

The system will support the design of graphic information presented on electronic displays. First the design is entered into the system using a graphics terminal. Next the movements and changes in response to input data are tested. Finally, the code which implements the design on the target display device is automatically generated.

In the design phase, the display designer will use graphic entities such as lines, circles, solids, or text to create pieces of the display design known as segments. He will assign attributes and dynamic controls to the segments, defining how and where they are displayed on the screen as a result of the various real-time system inputs. Segments may in turn be nested in other segments which move and change in relation to one another to form modules which make up the complete display design. These display designs, and portions thereof, may be stored in libraries for reuse. Throughout this phase, the designer will be supported by powerful editors which present the possible choices at each step--an environment optimized for the intermittent user.

The output of the design phase will be the formal display description. This formal description will be the basis for all other operations of the system. It will be totally comprehensive and unambiguous. It will be in a form such that a programmer may manually write the necessary code to implement the design on the target graphics hardware without needing any other information about the display design.

In the test phase, the designer will use simulated input data to animate the display design on the screen. This animation process will transform the formal display description into a sequence of rapidly changing frames. The designer will be able to specify screen update rates from frame-by-frame up to full speed, allowing him to identify such problems as incorrect spacing of elements or distracting use of color. If a fault is discovered, the display design can be modified and retested repeatedly.

After the display design is found to be acceptable, the formal display description will be transformed into the executable display description which uses the target hardware's instruction set to generate the images on the target display device. This is the code generation phase. The transformation may be done either manually by a programmer or automatically by the system.

2.2 System Partitioning

Figure 2-2 shows how the system may be best partitioned into major components to provide the needed capabilities. The notational convention used is as follows: closed boxes represent physical devices; open boxes represent data; circles represent processes which manipulate or change data; double circles represent persons interfacing with the system; and directed arrows indicate the flow of information from the sources of data to their destinations.

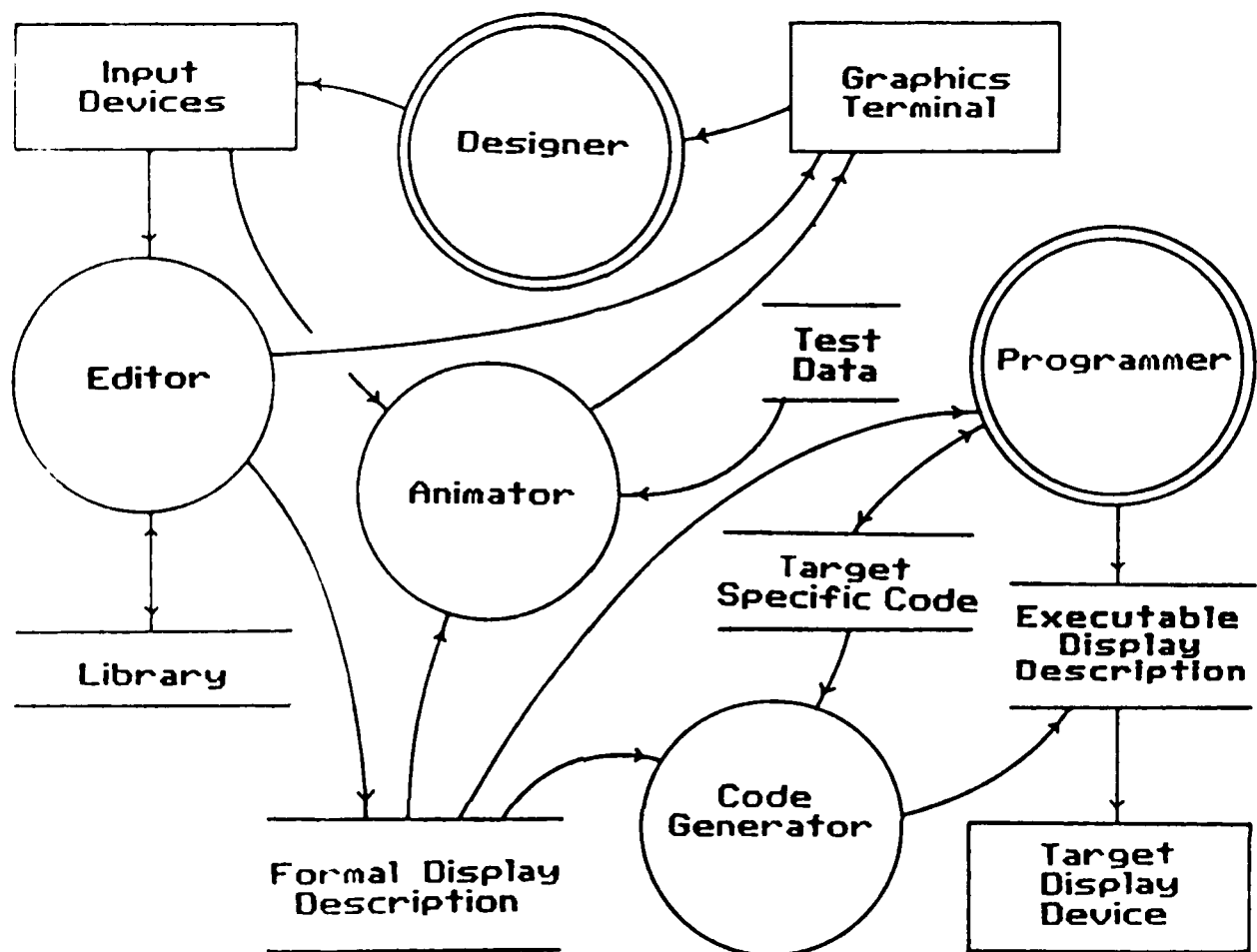


Figure 2-2 The System Partitioning

The data consist of the formal display description, the test data, the library of display components, the target-specific code, and the executable display description. The formal display description is a comprehensive and unambiguous description of the appearance of the final display. The test data are used to vary the display in such a manner as to allow the designer to evaluate the display design. The target-specific code is written by the programmer to support the creation of the executable display description. The executable display description is the executable program which may be run on the target display device.

The processes consist of the editor, the animator, and the code generator. The editor allows the designer to create the display design. The animator allows the designer to evaluate the design by seeing it change on the

screen in real time. The code generator automatically produces the program which will run on the target display device.

The physical devices consist of the input devices, the graphics terminal, and the target display device. The input devices allow the designer to give commands to the system. The graphics terminal allows the designer to observe the effects of his commands and to view the display design. The target display device is the equipment which produces the display in the real-time environment. Two other physical devices are not shown to simplify the drawing: the host computer on which the system is run and a hard-copy printer.

The persons involved in the creation of a display design include the designer and the programmer. The designer is the person with expertise in presenting information to the pilot or display viewer. His strengths are in human factors considerations, not in programming. He is responsible for designing and evaluating displays. Often, others without human factors expertise fill the role of the display designer. Some are responsible for building the target devices and wish to quickly design a display which will test various aspects of the device. For them, optimal displays are those which stretch the capabilities of the hardware. Rapid iterations may be needed to find the limits of a particular device. Other people who need to have input into the display design include representative display viewers who will work with the end product and programmers who can suggest improvements which allow more efficient generation of displays. In the context of designing displays, we will mention only the designer with the understanding that many people with a variety of backgrounds may be involved.

The programmer is a software engineer with at least an associate degree in computer science or equivalent experience. He may, but not necessarily, have a background in computer graphics. He is responsible for writing the subprograms which gather data from the sensors and other computers, subprograms which actually control individual pixels on the target device, and subprograms which define custom entities. These will be explained in detail below.

The system partitioning given in Figure 2-2 closely parallels the creation of display designs as described above. Each piece fulfills a need for creation of a particular types of data. The need for an unambiguous specification of the display is fulfilled by the formal display description, and the editor's sole purpose is to facilitate the creation of this formal description. The need for data specifying any necessary modifications in the display design is fulfilled by the moving images presented on the graphics terminal. The need for executable code running in the target display device is answered by the executable display description. Experience in software engineering has shown that when examination of the inputs and outputs of the system--the data--precedes examination of the processes, the final system is much more likely to fill the need.

The system partitioning is in terms of the functionality of each piece. It is an abstraction of the system that does not necessarily represent different programs or computers communicating to one another. It shows what components are needed in the implemented system at a high level in order to provide a solution to the problems of programming target display devices.

2.3 Data Requirements

One item of data is prevalent throughout the system: the display design. As mentioned in Section 1, the "display" is the time-varying pictorial mapping of information onto the target device screen. Specifically, the display is a sequence of frames drawn by the target device based on input data values, and the impression of motion is created by the frames being shown at a great enough rate of speed. Each frame in the display is composed of images, where images are the static visual representations of objects or groups of objects. The display design created by the designer is the abstract description of what images are displayed on the screen and how they are modified by the input data values from sensors and other computers. The abstract description is expressed concretely by the formal display description and the executable display description. This section contains the requirements for these three descriptions of the display. It also contains the requirements for all supporting data: the library, the test data, and the target-specific code. The formats of these data items will be specified during system implementation.

2.3.1 Display Design Requirements

The display design is an abstract, format-independent description of the contents of the display and how changes in the input data affect the mapping of visual information onto the screen. In order to facilitate the creation of usable, unambiguous displays, we have defined a hierarchical structure to be used by the designer in display designs. This section is divided into the following parts, reflecting the display hierarchy:

- o The Display Design Hierarchy,
- o Entities,
- o Segments, and
- o Modules.

2.3.1.1 The Display Design Hierarchy

The display design is composed of a module which is a collection of one or more segments. Each segment consists of entities, other segments, and module invocations. An entity is in turn composed of a collection of pixels. A pixel is the smallest resolvable area of a graphical display device and as such is the fundamental component of graphic images. This hierarchy is illustrated in Figure 2-3.

Items affecting the final appearance of the display are associated with different levels of the hierarchy. The association of items with particular levels is shown in Figure 2-4. This illustration will be briefly explained in the following paragraphs with more complete explanations given in the following sections.

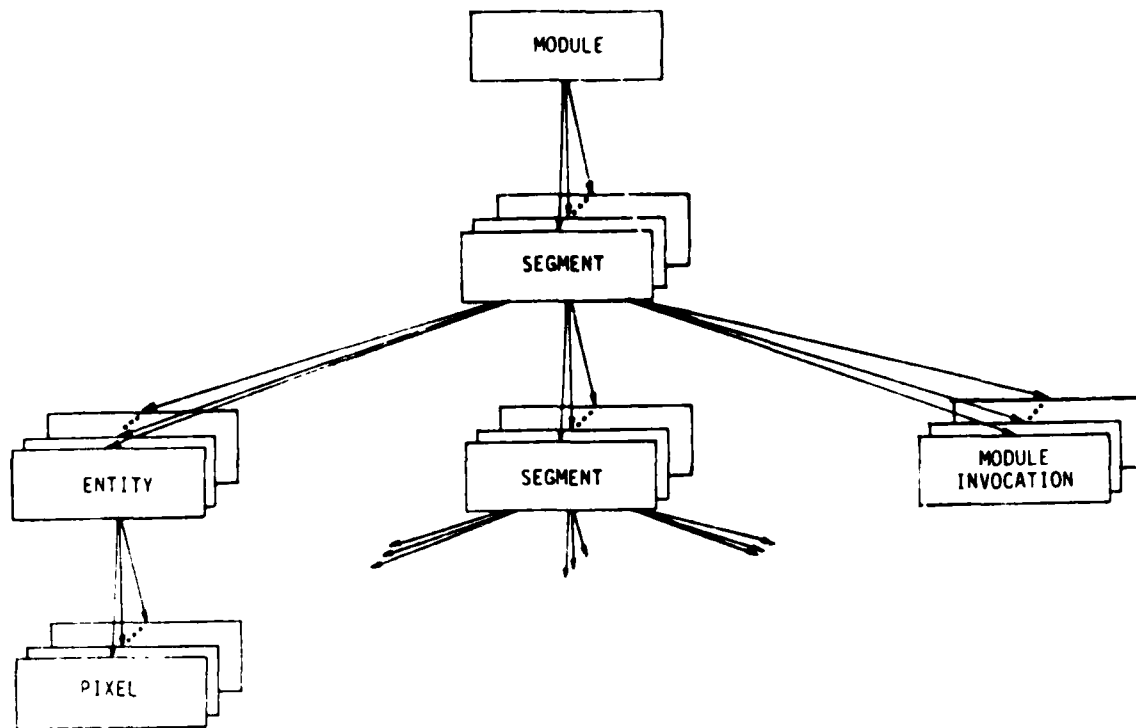


Figure 2-3 The Display Design Hierarchy

	NAME	PARAMETER SPECIFICATIONS	SERIAL NUMBER	CLIPPING BOUNDARY	COORDINATE SYSTEM	DYNAMIC CONTROLS	ATTRIBUTES
MODULE	X	X					
SEGMENT			X	X	X	X	X
ENTITY							X

Figure 2-4 Characteristics Associated with Hierarchy Levels

The realization of the display design is termed the 'display'. A particular display is defined by a single module. This module has a name associated with it, which is the name of the entire display. This module also names the input parameters of the display design. New parameter values transmitted from the sensor subsystems to the display subsystem will be reflected when the module is redrawn. As mentioned above, a module is composed of one or more segments.

These segments are in turn composed of zero or more entities, zero or more subsegments, and zero or more module invocations. Subsegments are segments which are nested in other segments. We will often refer to these as 'children' segments. A module invocation means that the invoking segment 'requests' that the named module be drawn on the screen. Also associated with each segment are a serial number, dynamic controls, and attributes. The serial number provides a means of identifying each segment and which segments are nested in others. Dynamic controls specify at what position and how the segment is to be drawn on the screen. Examples of such controls include rotation and translation along an axis.

An attribute is an item of information which describes the appearance of an object. When defining an object, the user defines both its shape and its appearance. Attributes are distinct from the object's shape. Examples of attributes include texture and color. Attributes may be defined at both the segment and entity levels. Definitions of attributes at the segment level will take precedence over those at the entity level when conflicts occur.

Entities are those objects which are seen as fundamental by the system. Although the only truly fundamental object in graphics is the pixel, the system needs to recognize more advanced entities such as lines, circles, and three-dimensional shapes. Thus the definition of an entity as used within this system is any graphical object which may not be subdivided to allow a particular attribute to be specified for one part of the object but not for another. Assuming a clock is not an entity, it may be divided into the hands and the face, with the hands colored red and the face colored green. However, a line cannot be partially yellow and partially white without actually being two lines. Attributes are the only items of information associated with the entity level.

When it is more convenient to the designer, a collection of entities may be termed an icon. Icons will be further explained in Section 2.3.4. Icons are not a part of the hierarchy and do not have any characteristics associated specifically with them. They exist only for the convenience of the designer.

The hierarchy and restrictions allow the designer to create displays which are structured rather than being an arbitrary collection of images at varying levels of complexity. Structure promotes ease of making changes in the display design because it encourages the grouping of related items. It allows the designer to postpone solving detailed problems. It facilitates the reuse of portions of display designs. It also allows the designer to specify and change which portions of the display are permitted to overwrite others on the screen. It allows the designer to control the extent to which commands affecting one portion of the display affect other portions.

To illustrate the use of the hierarchy to define a display, consider the simple display shown in Figure 2-5. This display consists of two objects which respond to the same input parameter. The first object is a linear scale with a moving slide whose position on the scale is determined by the current value of the input parameter, within maximum and minimum allowable values. The slide also contains the actual numeric value of the input parameter. The second object is a gauge which has been previously defined and stored as a standard gauge. The gauge contains a needle which rotates based on the value of the input parameter, to a position between the minimum and maximum allowable values.

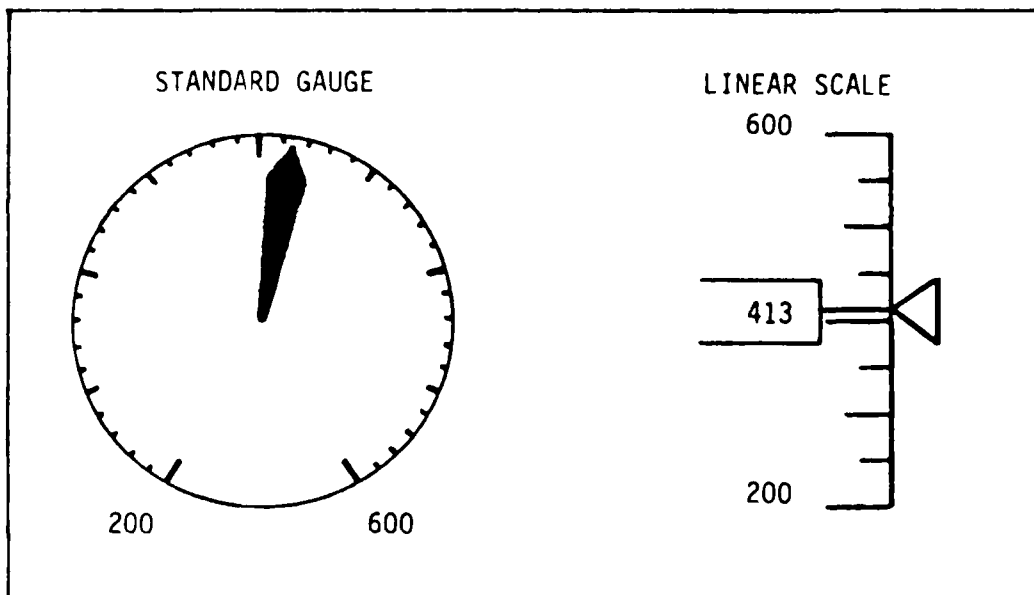


Figure 2-5 A Simple Display

Figure 2-6 shows how this display is constructed using the hierarchy. The entire display is a module which is composed of two segments: one for the linear scale and one for the gauge. The module specifies the input parameter to be used and passes the input parameter to the segments it contains.

The segment for the gauge consists of two major subpieces: the text of the gauge which is a collection of textual entities; and the invocation of the standard gauge module. The textual entities are the letters that make up

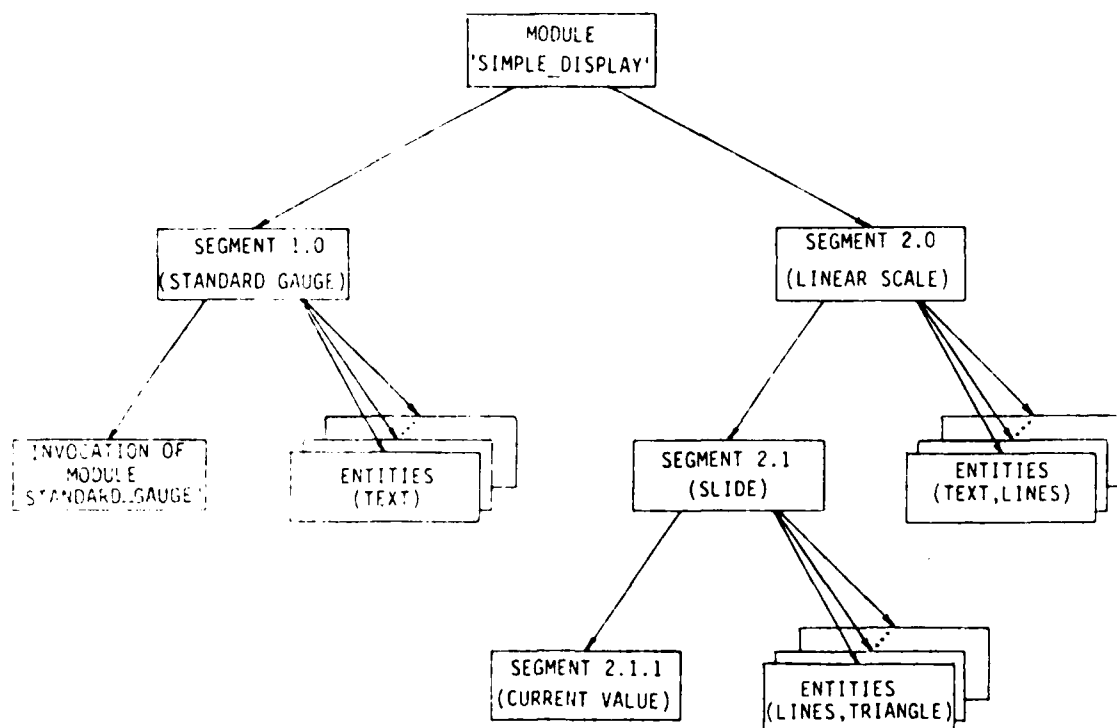


Figure 2-6 The Structure of the Simple Display Using the Hierarchy

'STANDARD GAUGE'. The standard gauge module is invoked by name, and the input parameter and maximum and minimum allowable values are passed to it during the invocation. These are used by the module to control the motion of the needle and to display the numbers '200' and '600'. The gauge segment specifies where the invoked module will be displayed on the screen using translations along axes and the size of the invoked module using scaling factors.

The segment for the linear scale is composed of several major subpieces: the text of the linear scale which is a collection of textual entities; the vertical scale which is a collection of line entities; and the slide, which is a segment. The segment also contains the dynamic controls to move the slide segment up and down based on the value of the input parameter with constraints to prevent it from moving off the scale. The textual entities are the letters and digits that make up 'LINEAR SCALE', '200' and '600'. The entities that make up the vertical scale are a vertical line and several horizontal lines of differing length.

The slide segment, whose position along the scale is related to the value of the input parameter, is composed of several entities and a segment. The slide box is composed of several entities including a triangle, a vertical line, and several horizontal lines of different lengths. The numeric value is in a segment whose content is a command to display the current value of the input parameter. As the segment which contains the numeric value of the input parameter is a part of the slide segment, the numeric value will appear to move up and down with the slide box.

This example shows the major aspects of the hierarchy and how they interrelate to facilitate the construction of complex displays. For a more thorough example showing the use of the hierarchy in constructing a display, see Appendix B. The characteristics associated with each level of the hierarchy will be discussed in greater detail in the following sections.

2.3.1.2 Entities

The following two-dimensional entity types need to be provided with the system:

- o points;
- o lines in various line styles such as dotted, dashed, or solid;
- o text, consisting of letters, numbers, and special characters;
- o regular polygons such as triangles, squares, and hexagons;
- o irregular polygons;
- o conic sections such as arcs and ellipses; and
- o irregular curves such as Bezier, cardinal, and cubic splines.

The designer needs to be able to fill any closed curve or polygon with any pattern. This allows such images as checkered backgrounds and colored schematics of vehicles. The above entities may be displayed in two dimensions. Support must be provided for three dimensions to allow depth to be incorporated for more realistic representations of objects and to transmit more information to the viewer. All two-dimensional entities should be usable in three dimensions. Also, regular three-dimensional entities such as pyramids, cubes, cones, and spheres are needed. The designer needs to be able to create irregular objects based on splines and other curves. Intersections and unions of these objects are needed to create holes in blocks or to build airplanes from rectangles and cylinders. Light sources in any color need to be provided so that shadows and highlights may be seen in three-dimensional scenes.

The designer needs to be able to depict three dimensional objects using either wire frames, panels, or continuous shading. In panel shading, the surface is approximated by a number of flat panel surfaces, and each surface is given a uniform color. While in continuous shading, a surface's color varies according to the angle at which light is reflected. It must be possible to map a texture pattern to a solid, molding it around the apparent body. All hidden surfaces need to be removed when displayed for clarity of images. Wire frames may have depth cues attached to them by the designers. Depth cues are scales which attach color or shade as a function of distance from the viewer so that lines in the background appear to be far away while lines up close are brighter. These scales need to be definable by the designer.

The system needs to support the creation of user-defined entities in two and three dimensions. This allows the designer to create images which are too difficult to build using the above set of entities alone. An example is a three-dimensional view of the terrain. Terrains could be drawn using large

numbers of general polygons positioned according to display parameters, but attaching the polygons to the parameters would be very difficult since each polygon must be placed precisely to simulate the actual world. Given that terrains will probably be used in many different forms under many conditions, highly flexible terrain entities must be supported. Since there is no well-known standard parameterization of a terrain, a predefined terrain entity would be useless in many situations. It is better to allow the flexibility of supporting user-defined entities than to attempt to foresee all entities which might be needed.

User-defined entities share the characteristics of the pre-defined entities, where pre-defined entities are those originally provided with the system as specified above. No command will allow an attribute to apply to one part of the user-defined entity without applying to the rest. It is possible that a user-defined entity will have changing attributes such as in a multi-colored terrain, but these colors will be treated as part of the entity itself, not as an attribute assigned to the entity.

2.3.1.2.1 Attributes

This section lists the available attributes. Again, attributes describe the appearance of an object. The following attributes will be supported by the system:

- o chromaticity,
- o gray level,
- o transparency, and
- o diffuseness.

These will be described in the following paragraphs.

The designer will be able to define the chromaticity of both the foreground and background. He will also be able to define the gray level of the foreground and background. Chromaticity refers to the combination of the hue and saturation of the color of an object, while gray level refers to the lightness or darkness of an object. We use the term 'color' to refer to the combination of these three aspects. Some target display devices specify color in terms of red, green, and blue components. The two systems are related in that changing a red, green, or blue component changes the chromaticity while changing all three proportionally changes the gray level. The user may work with whichever system is most convenient.

Many graphic display devices allow use of a certain number of colors at a time out of a larger range. This larger range is called a 'palette'. The range of possible colors in the palette should not be arbitrarily limited by the system because it is not possible to predict all of the colors which may need to be used in future displays.

It will be possible to specify the transparency of an object, i.e., to what extent an obscured item will be seen through a covering object. The range will be from total transparency to total opaqueness. Transparency could be used to show invisible objects such as the extent of a threat envelope or

the walls of a building. It could also be used to show shadows for three-dimensional objects by defining the shadow as a transparent gray shape. If the transparency is below a given threshold, covered lines will be dashed instead of solid (unless the object is totally opaque). Dashed lines are thus used to distinguish between hidden objects even though the colors are no longer distinct. This threshold will be specifiable by the designer. Transparency needs to be cumulative in such a way as to give realism when images are viewed through multiple semi-transparent objects. For instance, if a mountain is behind overlapping, transparent threat envelopes, an aircraft pilot would like to be able to see the mountain behind the envelopes.

The diffuseness of an object needs to be specifiable, where diffuseness is how much of the light striking an object is scattered back to the viewer. Diffuseness will range from being totally specular to totally diffuse. This applies only to three dimensional objects in scenes with one or more light sources. Diffuseness lends realism to a scene by showing the roughness or smoothness of objects.

Like transparency, diffuseness needs to be cumulative for multiple reflections. However, the extent to which multiple reflections are shown is not as important as other aspects of the images on the screens since only partial support of multiple reflections may not be noticeably different from full support.

As mentioned above, attributes may be defined at both the segment and entity levels. By allowing entity-level definitions of these attributes, the designer can easily create complex, multi-colored objects without using large numbers of segments, one for each color. By allowing segment-level definitions, the color or transparency of the entire object may be changed with only one command. However, if the designer wants some parts of an object to change to blue on command and other parts to change to green, each part will need to be defined in a separate segment.

2.3.1.3 Segments

Segments are collections of entities, icons, subsegments, module invocations, dynamic controls, and attributes. Each segment needs to also have a coordinate system and a clipping boundary specified by the designer.

All movements of the segment as specified by the dynamic controls will use the specified coordinate system. In the context of this report, coordinate system refers to the measurement units, the placement of the origin, and the orientation of the x, y, and z axes. Each segment may have a different coordinate system. The designer will specify how the coordinate system of a subsegment relates to that of its parent. The segment may only move as a whole; the parts of a segment are placed relative to each other in the same way each time the segment is redrawn. The coordinate system allows the designer to specify what point will be the origin in all transformations. If a coordinate system were not present, the effect of transformations would be ambiguous.

The designer shall also be able to specify a clipping boundary which is delimited by any closed two or three dimensional shape. The edges of the clipping boundary define a border across which no portion of a segment may be drawn. This boundary can be delimited by any shape so that the designer has full flexibility. Clipping boundaries allow different segments to overlap each other without losing clarity. Using clipping, an instrument panel made of line drawings placed over a three-dimensional map of the terrain can be clearly distinguished from the map by giving the instruments a black background and a clipping boundary. The alternative would be to attempt to use different colors for all of the lines so that each may be distinguished. This may be impossible in certain situations, especially on monochrome target display devices. Clipping is shown in Figure 2-7 in which boundaries are shown with dashed lines. The actual clipping boundary for a nested segment (e.g., Segment 3.1.2 in Figure 2-7) will be defined by the intersection of its parent segment's (e.g., Segment 3.1 in Figure 2-7) active clipping boundary with its own predefined clipping boundary. This new clipping boundary becomes the active boundary for all children of the nested segment (e.g., Segment 3.1.2 in Figure 2-7). Clipping boundaries, like entities, will remain static in size, shape, and position relative to the coordinate system of the segment.

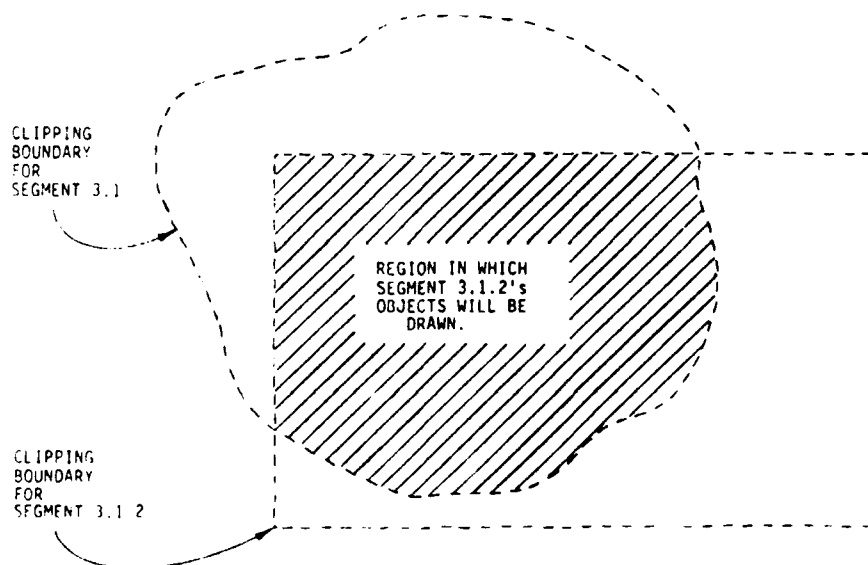


Figure 2-7 The Intersection of Clipping Boundaries

As segments may be composed of other segments, it must be possible to nest segments inside each other to practically any depth. To allow the designer and the editor subsystem to keep track of which segments are nested in others, each segment will have a unique serial number. The 'highest' segments, those which are directly contained within a module, will have serial numbers of 1.0, 2.0, 3.0, and so on in whatever order is appropriate for the particular display. The children of a segment (those which are nested within that segment), will be numbered using the parent's number as a prefix in the same way sections are numbered in this report. As an example, the three children of 8.4.5 would be 8.4.5.1, 8.4.5.2, and 8.4.5.3. In general, if S is

the serial number of a segment with n children, they will be numbered $S.1$, $S.2$, ..., $S.n$. This is illustrated in Figure 2-8.

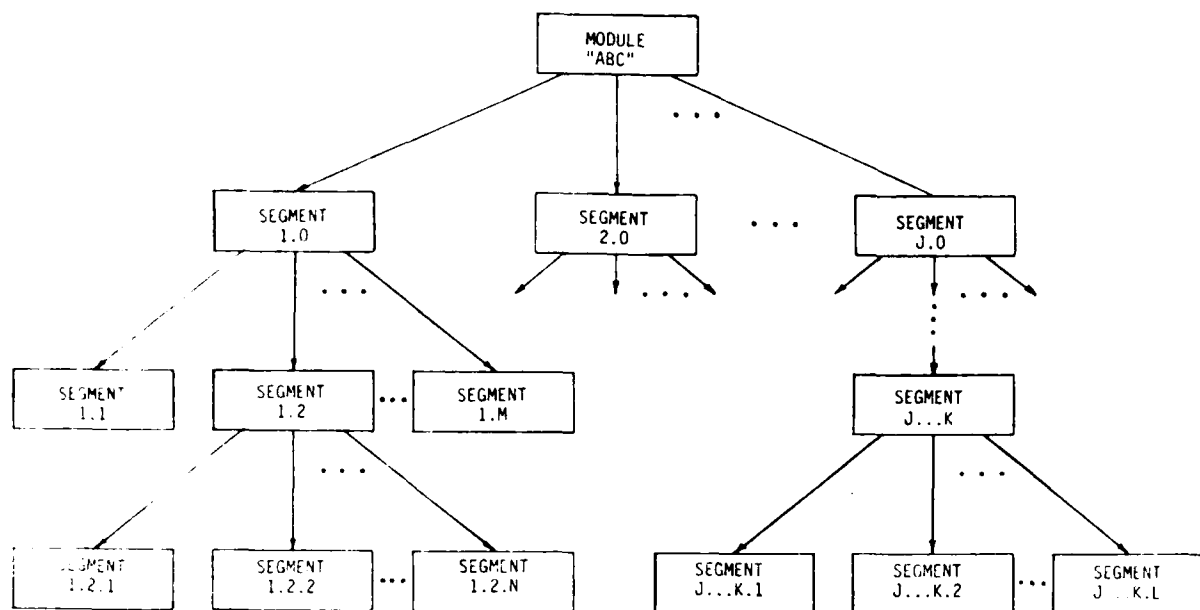


Figure 2-8 The Segment Numbering System

When a display is to be redrawn, the numbering system is used to determine the priority of segments where priority expresses which segment is to be showing when two overlap. Segments with higher priorities will appear to be drawn on top of other segments. The serial numbering system expresses which segments have the higher priorities where the ordering is such that all serial numbers beginning with 1 are drawn before all those beginning with 2. Likewise, all those beginning with 1.1 are drawn before all those beginning with 1.2, and so on. Also, each parent segment is drawn before its children. Thus segment 2.0 and all of its children will have a lower priority than any of the segments with serial numbers beginning with 3. This ordering is illustrated in Figure 2-9 where letters are used to show the priority. The segment indicated by the letter "a" has lower priority than the segment indicated by the letter "b", and so on.

2.3.1.3.1 Dynamic Controls

Dynamic controls are used to specify where on the screen an object defined as a segment is to be drawn. They are also used to control whether a given segment is to be drawn or updated in a given frame. Finally, they are used to specify how numeric or textual information defined as a segment or subsegment is to be presented to the viewer. Dynamic controls will be provided for the following items associated with graphics:

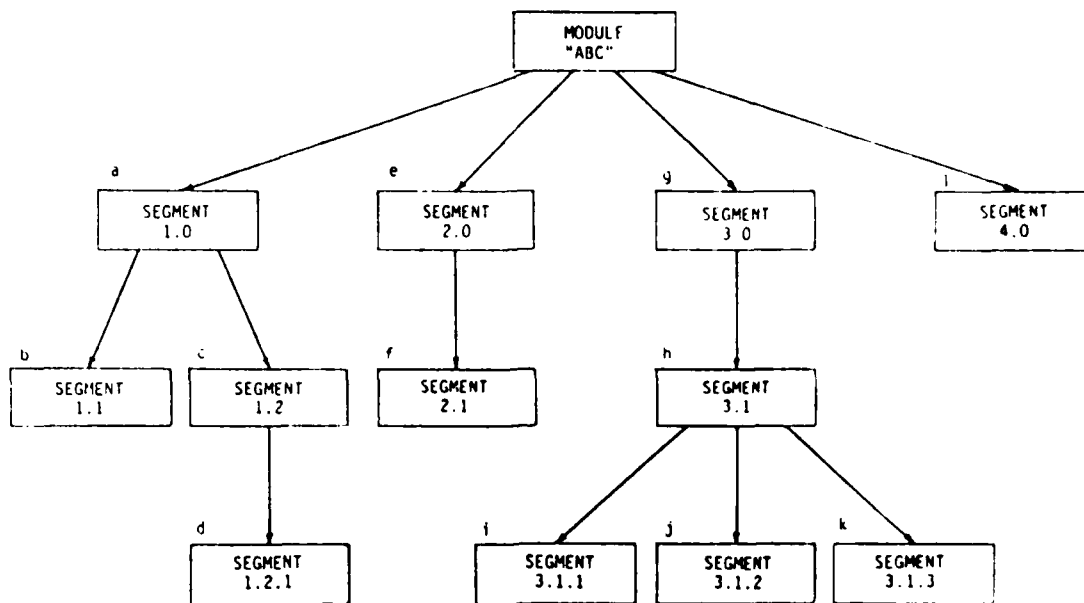


Figure 2-9 The Priorities of Segments

- o the scale along any axis,
- o mirroring about any axis,
- o rotation about any axis,
- o translation along any axis,
- o location of the viewpoint relative to the axes,
- o direction of view,
- o twist along the direction of view,
- o distortion of perspective,
- o field of view,
- o drawn or not drawn,
- o blinking rate,
- o priority, and
- o clipping method, and
- o update rate.

Dynamic controls will be provided for the following items associated with both textual and numeric data:

- o height,
- o width,
- o font style,
- o relative angle,
- o field width,
- o justification within a field, and
- o character used to pad a field.

Dynamic controls will be provided for the following items associated with numeric data only:

- o use of signs,
- o use of commas,
- o use of decimal points,
- o number of decimal digits displayed, and
- o type of exponential notation.

Scaling, mirroring, rotating, and translating a segment in response to input parameters allow the designer to make an object appear to move on the screen. Each time a segment is drawn, the system will recalculate its position given the equations specified by the designer. For instance, a clock hand might be made to rotate according to the equation $[90 - (30 * \text{HOURS}) \text{ degrees}]$ so that when the value of HOURS was 12 the clock hand would be vertical and pointing upward, and when the value was 9 it would be horizontal and pointing to the left.

All positional changes are relative to the segment's coordinate system. This coordinate system's orientation relative to its parent is specified by the designer. All transformations are to take place in the order in which they are specified and are to be cumulative: rotating and then translating a segment may be different from translating before rotating as shown in Figure 2-10. This allows the designer to specify segment movements in whatever way is easiest for the particular situation.

The transformations listed above refer to moving the segment on the screen. It will also be possible to specify the direction from which a three-dimensional segment appears to be viewed by specifying the location of the viewpoint, direction of view, twist, distortion, and field of view. The first defines the viewer's apparent location relative to the segment. The next specifies what the viewer is looking at. Twist allows the segment to appear to be at an angle or upside-down by twisting the viewer's horizon. Field of view defines a cone along the direction of view so that all objects falling within the cone will be displayed. Perspective distortion allows exaggeration or suppression of apparent distances between parts of objects.

The designer needs to be able to control whether or not a segment is drawn. If a segment is not drawn, all subsegments of that segment will not be drawn. This allows portions of the picture to be conditionally displayed.

The rate of blinking of a segment will be specifiable. This will range from as fast as possible to no blinking. Blinking can be helpful for drawing attention to important images in the display.

The designer will be able to make use of user-specified priorities to override the default priority system based on segment numbers. Use of these priorities is demonstrated in Figure 2-11. This figure is based on Figure 2-9 with the letters indicating priorities as before, except that the user-specified priorities are taken into consideration. User-specified priorities may be used to re-order the priorities of sibling segments and their subsegments. They may not be used to re-order the priorities between the children of one segment and the children of another. For example, in Figure 2-11, Segment 3.1.2 could never have a priority between the priorities of Segment 1.2 and Segment 1.1. User-specified priorities allow different parts of the display to be interwoven, providing greater flexibility to the

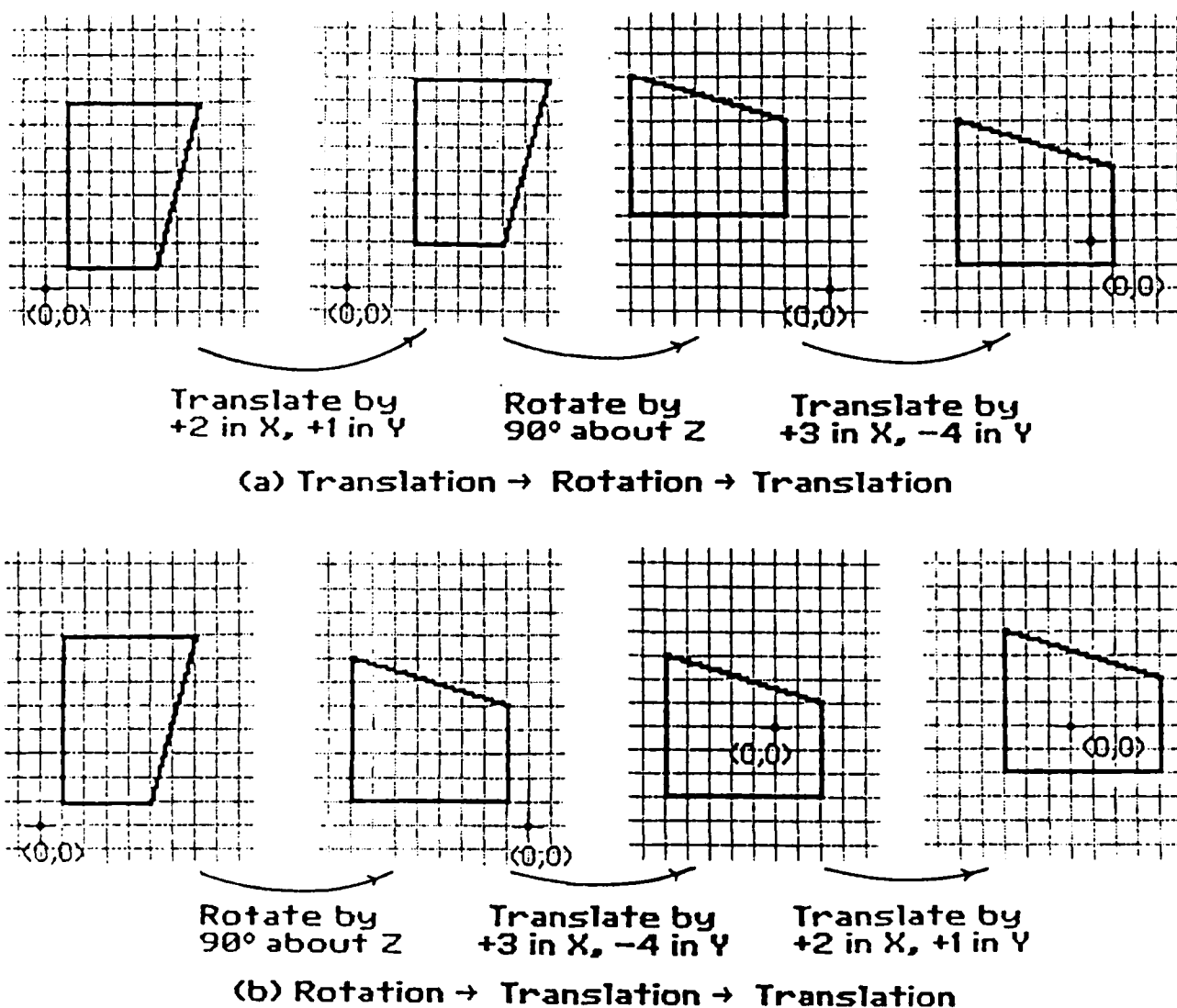


Figure 2-10 The Cumulative and Noncommutative Nature of Transformations

designer. Furthermore, the designer may change the priorities so that objects on the screen may alternately be hidden or revealed.

These user-specified priorities will be specified numerically with higher numbers representing higher priorities. Higher priority segments are drawn after, or in front of, lower priority segments. If two segments are given the same user-specified priority, the default priority system based on the segment numbers will be used to determine the priority as shown by segments 2.0 and 4.0 in Figure 2-11. All segments without user-specified priorities are assumed to have a lower priority than any sibling segments which have been assigned user-specified priorities. This is shown by segments 3.1.1, 3.1.2, and 3.1.3 in Figure 2-11.

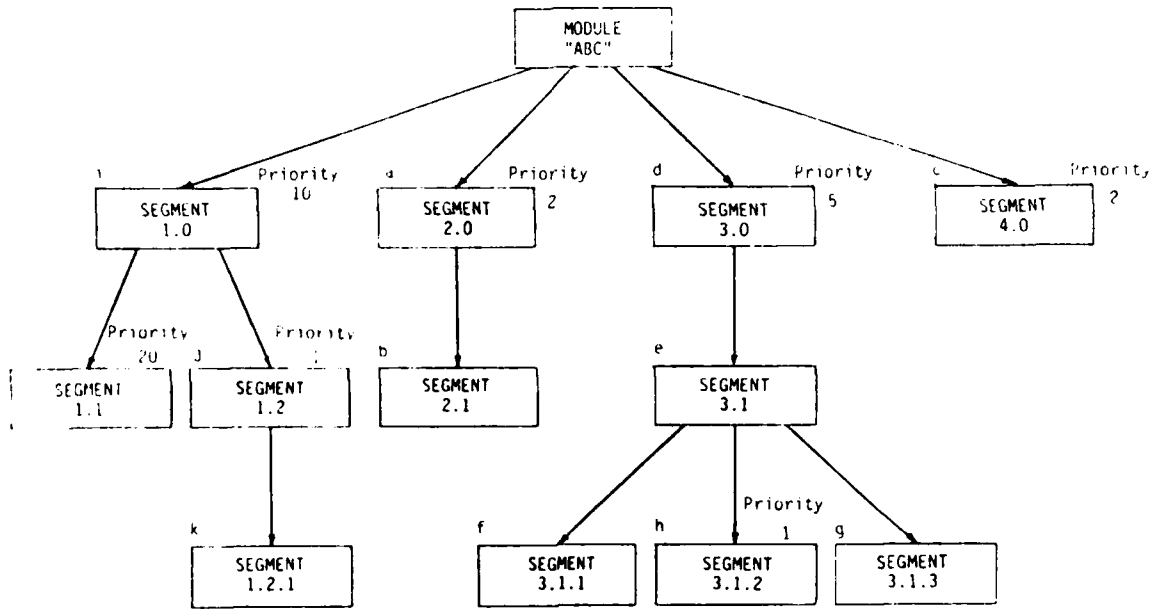


Figure 2-11 The Use of User-Specified Priorities

The method of clipping a segment will be specifiable as none, partial, or whole. No clipping means that the clipping boundary will be ignored and the system will assume all lines will never be drawn outside of the specified limits. This especially applies to segments which do not move at all. Specifying no clipping could increase the update rate for the display by allowing the computer to ignore the possibility of the segment stepping beyond its bounds. Partial clipping means that whenever part of a segment strays beyond its bounds, only that part will not be displayed--the rest of the segment will be displayed. This is the most common clipping technique. Whole clipping means that whenever any part of a segment would be clipped, the entire segment (and all subsegments) will be suppressed. This is often used to suppress messages which will not fit on the screen. It may also be used to speed the update rate by simplifying the processing which must take place for each frame.

The designer will be able to specify the update rate of a segment from a given range of values, where update rate is the rate at which a portion of the screen is redrawn with a new frame. This is not to be confused with the rate at which the hardware refreshes the images on the display device. This can be used to increase the update rate of the rest of the display by allowing the computer to ignore updating a particular portion of the screen. It may also be used to slow down how often a particular portion is updated so that it does not move too quickly for the person viewing the display. For instance, the viewer may be distracted by the rapidly changing digits of a digital altimeter which is updated 40 times a second, and would prefer to see it change only twice a second.

For all textual and numeric data, the designer will be able to specify the relative angle, height, width, and font style of the characters used to display the data. Relative angle here refers to the angle at which each character is drawn with respect to the base angle of the entire text. For example, italicized text may be created by slanting each character a few degrees from the perpendicular. Height controls how tall each character is displayed, while width controls the thickness and relative spacing. Specification of the above controls allows emphasis to be placed on important messages or to improve the readability of text.

Again for all textual and numeric data, the designer will be able to specify the minimum width of the field which the data will occupy on the screen. The field will be oriented in such a way that the lower left corner is positioned at the origin of the segment, and in such a way that it is parallel with the surface of the screen so it is readable. A specifiable character will pad the field so that the data is either left justified, right justified, or centered by the designer. These controls allow clear presentation of textual information which is easy to read.

In addition, for numeric data, the designer needs to be able to specify the format of the data: whether and where signs will be displayed to denote positive or negative values, whether commas are to be used, how many decimal digits are to be displayed after the decimal point, and whether exponential notation is to be used. These controls allow clear presentation of numeric information so that it is as readable as possible.

As an example of the differences between shapes, attributes, and controls, consider a display which depicts an enemy aircraft in three dimensions. The body, wings, and fins of the plane would be formed from various entities such as cylinders and triangles. Attributes would be used to make it smooth, highly reflective, and silver in color with the appropriate markings. The transparency attribute would be used so that one could see the pilot in the cockpit for added realism. All of these define the airplane object.

Given the object, the dynamic controls are used to position it on the screen so that it appears to be in the correct position in the sky relative to the viewer's plane. There are two ways this could be done. One would be to offset the enemy aircraft from the viewer's position through a sequence of rotations and translations. Another technique would be to specify viewpoint coordinates relative to the enemy aircraft's coordinate system. Which way would be preferable for a particular application depends on the type of parameters received. Whichever technique is used, as the parameters are updated the enemy aircraft will appear to move against the background sky or ground.

There will be a separate list of graphic controls for each segment to specify the above transformations. If two segments are to be moved together, they should be combined into one or both be made children of a parent segment in which the controls are placed. Regardless of the form in which they are specified, graphic controls need to also allow the designer to designate whether a particular segment is viewable or not. They must allow specification of the invocation, or drawing, of submodules--modules which are used as part of a segment. It must be possible to pass data to these modules

from the segment. Dynamics must provide conditional control of a segment so that lists of actions take place only under certain situations. They must allow conditional looping so that lists of controls are acted upon repeatedly. Dynamic controls will be used by the designer to display messages and data on the screen, such as a warning message or the numeric value of a temperature gauge.

The final note concerning dynamic controls is that they will not accumulate between frames. If in one frame an object is moved to the right 2 inches and in the next 3 inches, the total movement in the second frame will be 3 inches, not 5. This allows the designer to ignore previous positions when specifying the dynamic controls for a segment. Since it would be useful to be able to base positions on previous segment positions, access to these values must be provided. One possible mechanism for this would be to provide such values as PREVIOUS_X_LOCATION and PREVIOUS_X_ROTATION. This allows the designer to move an object across the screen without needing an input specifying elapsed time.

2.3.1.4 Modules

Each module will have an alphanumeric name by which it may be referenced. Each module will have a set of specified input parameters. Each input parameter will be given a name by which it may be used. For example, a module used to show direction might have the input parameter names will be used by the segments within the module in expressing dynamic changes.

Within a segment of a module, it will be possible to evaluate expressions formed from the named inputs and from function calls with the inputs as parameters. For instance, $\text{LOGARITHM}(\text{TEMP} * 2 - 10)$ would be a legal expression, assuming that TEMP is greater than 5. The functions which can be called will include the following:

- o trigonometric functions (such as sine, cosine, tangent, acrsine, arc-cosine, and arctangent),
- o other transcendental functions (such as exponential, logarithmic
- o conversion functions (such as rounding, truncating), and
- o designer-defined functions.

If an expression within a segment is illegal, such as when dividing by zero, the system must handle it gracefully. One possible technique would be to display a message on the screen informing the viewer of the error and the possible corruption of the data in that frame.

Input data will come from the environment outside the display subsystem. These data may also be received as specified below from other modules. Input data may be either single items or lists of items. For example, "CURRENT_HEADING" would be a single input while "ENEMY_POSITIONS" would be a list of inputs.

One module will be able to invoke another by name from a segment within the invoking module. This will cause the invoked module to draw itself on the screen. Hence, an instrument panel may be composed of several modules invoked from, for example, Segment 5.3.8. Only modules, not segments, may be invoked in this manner; the segmentation of a module is hidden outside of that module. An invocation results in the contents of the invoked module being displayed as if it were a subsegment to the segment from which it was invoked. An invoked module will be able to receive data from the segment. The system will support the passing of the data by allowing the designer to state which parameters in the invoked module are given which new values. This invocation allows: the drawing of a module in a frame more than once, such as for a fleet of ships; the structuring of displays into well-defined components; and the one-time definition of a component which may be used in many display designs.

2.3.2 Formal Display Description Requirements

As mentioned in the discussion of the need, a primary problem in the current method of creating displays is the lack of communication of display designs. An unambiguous description is needed to solve this problem. The formal display description fills this need. It is formal in the sense that it does not use a natural language for the description. While English can be unambiguous, being so is not a necessary part of a statement in the language. If it were a necessary part of the language, no sentence would be misunderstood by any knowledgeable person. The formal description will be defined in such a way that it cannot be misinterpreted by any part of the system, including the designers. Thus the formal description will be an absolutely unambiguous description of the content of the display and its response to test data.

The formal description will be readable by a programmer so that he may use it to program the target hardware manually when it is impossible or undesirable to do so automatically. All displayed images will be specified from the symbolic description level down to the level of pixels drawn on the screen so that the programmer does not need any knowledge about graphic algorithms.

One possible way to formally describe the display would be to provide a bit map definition of the display. A bit map is a matrix in which every pixel on the screen is given a corresponding color. Thus a line would be stored as a bit map with the values in most of the bits representing the background color and a small number representing the color of the line. This is unambiguous because one knows precisely which pixels will have what values.

Bit maps, however, provide very little abstract information to the programmer. He must have a symbolic description of the display to be able to program the target display device. The format of computer code is very different from a bit map, and using bit map definitions would add an extra step in the process by making the programmer determine the types and locations of shapes from the bit maps. If the information can be transmitted in a more readable form such as in the statement "draw a line from (20,3) to (80,80)," determining the types and locations of objects in the display is simple. Thus, if a line is to be drawn on the screen, it will be specified as a line

so that the programmer's job is simplified. This level of detail allows a programmer to manually write the code to run the target device. He should need to do as little interpretation of the formal description as possible while creating optimal code.

As mentioned above, it is assumed that the programmer has a very limited background in computer graphics. Thus all algorithms which are referenced within the formal description should be given to him along with references to alternative algorithms. This includes algorithms for both graphics (such as those for continuous shading) and mathematics (such as those for tangents and logarithms).

2.3.3 Executable Display Description Requirements

The purpose of this system is to place displays in the target display device. It would defeat the purpose of the system to a certain extent if the only output was a formal description of the display because further transformation is needed before the target device may execute the display. Hence the system will create the executable display description. This will be the code which may be transferred to the target hardware. It will instruct the target hardware on how to draw the display design in real time, in excess of 50 frames per second. The difference between the executable and formal descriptions is that the latter is in a general form while the former is for specific hardware. The data will also be in whatever format is necessary for system implementation. It can be created either automatically by the code generator subsystem or manually by a programmer.

2.3.4 Library Requirements

To solve the need for reuse of display components, the system will maintain a library of these components. The stored components will be of several types: user-defined entities, icons, segments, and modules. Each stored component will be given a name by the designer so that it may be retrieved using that name at a later time.

Modules are stored in their entirety, complete with the defined inputs and segments. Example modules would be a compass or a truck. When the designer specifies that a module will be incorporated into the display at a certain place, the editor will show a static image of the module in the display upon command so that the designer can determine if its use is correct. However, only the name of the module will be placed in the formal display description, not the module's full description. The module will be referenced only by name until the last possible moment, at which time its description will be inserted in the display design. Thus if a display makes use of a module, any changes in the module will be reflected in the display whenever that display is regenerated. This allows several designers to maintain consistency of images used in displays.

Segments are stored in libraries with their associated attributes and dynamic controls as well as the entities, icons, subsegments, and module invocations which form the segment. Examples of stored segments might include a row of dials from an instrument panel or a generic wing of an airplane with movable flaps. Changes to library segments do not change the displays in which those segments have been previously used. When a display is built and the designer incorporates a library segment, the description of that segment is immediately inserted into the display design.

Icons are groups of entities with their attributes which are often used in displays. The needle of a compass or a symbol representing a hospital are typical icons. Icons, like segments, are incorporated during the building of the display. If a library icon is changed at some later time, any displays into which the icon had been previously placed will not change. However, any display designs built after changing a library icon will reflect the changed version of the icon.

2.3.5 Test Data Requirements

To allow the designer to evaluate the display design, test data are needed so that the animator may simulate how the display will appear in the final environment. The test data will simulate inputs which are to be given to the target hardware by the data-acquisition computer shown in Figure 1-1. The designer will be able to generate the test data from a variety of sources. To test 'normal' conditions, data could be generated from taped recordings of the actual environment or by test data generator programs which simulate the sensors and other computers. To test 'impossible' conditions, test data could be generated manually by the designer.

2.3.6 Target Specific Code Requirements

The target specific code is data generated by the programmer to support the automatic code generation process. The programmer uses the target device's instruction set in creating the necessary target specific code for a given application. Where the formal display design defines the display down to the level of pixel actions, the target specific code states how instructions are used to manipulate the pixels in a particular target. This target specific code also states how data will be received by the target display device from the data-acquisition computer.

Since the target devices are constantly being upgraded to include new technology, the instructions used to generate an image will differ from target to target. Also, the order and format of the data values sent from the data-acquisition computer will depend not only upon the computer but also upon the application. Hence, this information must be provided by someone familiar with the instruction set and characteristics of the hardware in use at a particular facility. This person should be a programmer because the documentation which describes the computer hardware is meant for use by those with a programming background.

2.4 Process Requirements

These are the requirements for the processes shown in Figure 2-2. These processes are the programs which transform the various pieces of data. The editor transforms the designer's commands into a formal description of the display. The animator draws the display design on a graphics terminal, updating the display in response to test data. The code generator transforms the display into the executable display description for use in the target hardware.

We list below those capabilities needed for the creation of optimal display designs. We will also specify in a general way how those capabilities are presented to the designer. It must be very simple to use the system. Furthermore, potential designers are often intermittent users, and ease of use is needed so they do not need to relearn the system every time a new display is to be designed. To support these needs, the system must make extensive use of menus and provide on-line help.

2.4.1 Editor Requirements

'Editing' is the process of creating and modifying a document or data file, in this case the formal display description. In this system, the editor will automate the process of interactively creating formal display descriptions. The editor requirements must facilitate the creation of any display which the designer would need to design. This section is divided into the following parts:

- o General Editing Capabilities, and
- o Default Attribute Values.

The first section lists the general needs associated with editing displays. The second section defines default values for the attributes of images in display designs.

2.4.1.1 General Editing Capabilities

These requirements allow the designer to create useful display designs with a minimum amount of effort.

The designer will be able to save, retrieve, and modify the above display designs including both the graphical images and the dynamic specifications. This is similar to the capabilities of document creation systems as mentioned above.

To allow the designer to modify the images as easily as possible, the editor must be capable of moving the images on the screen. This movement is not to be confused with the movement observed during the animation phase of the design process; it refers only to the temporary repositioning of objects during the edit phase. Thus the designer may draw a picture on one face of a cube and then turn the cube to draw a picture on the opposite face. Whereas

textual editors only need to provide the ability to move a window forward and backward through a file, the graphics editor must support movement in all directions. The editor must also provide the ability to view multiple windows simultaneously and to move between windows easily. This is necessary to allow the designer to view various portions of a display at the same time or to preview a library component for possible inclusion in a display.

The editor must support the hierarchy of displays as defined above. The designer needs to be able to define modules and segments within those modules with all of the necessary associated information about parameters and dynamic controls. This does not mean that the editor will allow the designer to see the dynamics while in the editor; he must use the animator subsystem to see the display actually change in time.

The editor will allow the designer to assign names to segments in a display during the editing process. These names exist only for the convenience of the designer. They will allow the designer to refer to a segment using a more mnemonic technique than the numbering system provides. Unlike the segment numbering system, names are not part of the design hierarchy. Both segment names and numbers are not accessible outside of the module in which the segments are defined.

Furthermore, the editor must allow the design of icons and other display components for placement in the library. The editor must allow the designer to maintain libraries, adding, deleting, and modifying items at will. Any library icon may be incorporated into the design with any scale, orientation, or segment membership. Segments may be incorporated as children of any other segment. Where the parameters of an incorporated segment are undefined in the current module, the editor should enforce their definitions.

To further support the library of components, the editor must allow the library to be organized using the concept of nested libraries, or sublibraries. Using sublibraries, the designer could, for example, have a library of all gauges with sublibraries for altimeter, temperature, and fuel level gauges. The editor must allow the designer to view both the names of the components in a library and the components themselves.

The editor will also assist the designer in designing a display according to the capabilities of a particular target display device. By entering the name of the target, the editor shall set up the screen so that there is a one-to-one mapping between the apparent size of the target on the graphics terminal and the target screen's actual size. Also, the palette of colors will be limited so that they correspond to the target display's palette when that palette is smaller.

If the screen is to be 4 inches by 8 inches the editor will block out all but a four by eight portion of its screen. This allows the system to transfer the display design to the size and shape of the target device. The designer shall then be able to expand or contract the display design so that he can examine in detail a particular area of the display. Such an expansion might mean that a portion of the display is beyond the borders of the screen. Thus the designer might be able to reposition the display on the graphics terminal to examine different portions of the image. While the display is so expanded, the effect of all commands expressing the sizes of icons and other

parts of images will be scaled appropriately. If the display is two times larger than normal, a line 2 inches long will appear to be 4 inches long. These capabilities allow the designer to position images precisely.

Constraining the colors used in the display allows the designer to be certain that the display design will appear on the target display as desired. Also, since many graphic hardware devices support a large palette of colors of which only a few may actually be on the screen at any one instant, the color constraint would limit the number of different colors in the display simultaneously. Whenever too many colors are used, a message will warn the designer of the problem.

The above constraints must be removable. Often the designer wishes to create a hypothetical display which is not limited by the capabilities of any particular target device. Then he must have access to the entire screen and palette of the graphics terminal. Furthermore, it must be possible to allow the designer to define constraints for any new target devices which might become available without necessitating a rewrite of the editor code.

2.4.1.2 Default Values

It is desirable to allow the designer to not need to specify every detail of a display when many of the details remain the same in the majority of cases. Defaults are values which are automatically given by the system in lieu of specification by the designer. Also, defaults are provided so that the display acts reasonably when information is not supplied. These defaults are specified by the editor when the displays are originally created as opposed to being added later by some other part of the system.

Below are suggestions for the default values. The implemented system should allow the designer to respecify the defaults at will. Thus where we specify that the default for a monochrome display is light on dark, he may change this default to dark on light.

2.4.1.2.1 Default Attribute Values

The default color for monochrome displays will be light on dark, while the default chromaticity for multi-colored displays will be white on black. The default gray level will be the median gray level of the gray scale being used. These decisions are arbitrary, and hence they underscore the need for respecification of defaults by the designer.

The default transparency will be opaque. This reflects the normal characteristics of objects in the world.

The default diffuseness will be halfway between totally diffuse and totally specular. Again, this reflects the normal characteristics of objects in the world.

2.4.1.2.2 Default Dynamic Control Values

The default clipping boundary will be the border of the screen. This allows maximal use of the screen space.

The default blinking will be no blinking. In most cases, the designer will not need to have objects blinking on and off.

The default priority will be that the segment with the higher serial number will have the higher priority as specified in Section 2.3.1.3.

This is provided for the purposes of reasonable behavior when the priority of a segment is undefined.

The default clipping will be partial clipping. This is the most widely used clipping technique.

The default update rate will be as many updates per second as supported by the hardware. Normally, the designer will want to have the display respond as quickly as possible to environmental conditions.

For text and numeric data, the default will be displaying all letters and digits horizontally using the display device's standard font style, height, and width. This font style is usually the one which is displayed on the screen the fastest.

For textual and numeric data, the default will be displaying the data in a field which is as small as possible. The field width should normally be defined by the designer, but a default needs to be provided so that the display behaves reasonably when the width is undefined.

For numeric data, the default will be that if the number is negative, it will have a leading minus sign; otherwise, no sign will be displayed. If the data are greater than the precision of the display device's integer arithmetic, it will be displayed using the exponential notation. Commas will not be supplied by default, and as many decimal digits as are within the precision of the machine will be displayed. Again, these defaults are provided for the purposes of reasonable behavior by display designs.

2.4.2 Animator Requirements

The animator uses both the formal display description and the test data to allow the designer to test his display design by moving and changing the images on the screen in real time. This is like a simulator in that it 'simulates' how the display will appear dynamically on the target device. Essentially, the animator repeatedly displays static frames of images at such a speed that the human eye perceives continuous motion.

The designer needs to be able to control the rate at which frames are drawn on the screen of the graphics terminal. He needs to be able to see the animation process happen at full speed--the same speed as the display would appear on the target device, at any reduced speed, or on a frame-by-frame

basis. These different rates allow the designer to carefully evaluate his display, perhaps identifying problems only partially noticed at full speed. At any time he will be able to pause, restart, skip forward, skip backward, change the speed, or abort the animation process. He will also be able to request that the data be read in reverse instead of forward. These different rates and techniques give the designer full control of the animation process.

While the animator is in the pause state, it will freeze the gathering of test data so that no test data are lost. While the animator is running at full speed, intermediate values of parameters which are generated faster than the frame update rate are ignored. However, when the animator is running at a reduced speed the intermediate values will not be ignored; a frame will be generated for each set of data values.

While the animator is in the pause state, the designer will be able to manually enter the values of parameters so that he may test 'impossible' conditions. He will be able to assign one or more parameters to a variable input device, such as a rotary or slide potentiometer, so that he may incrementally change their values and see these changes reflected on the display. The designer will be able to increase or decrease a scaling factor which controls how much change in the input device is needed to produce a proportionate change in the parameter. If desired, the designer will be able to continue to use the variable input device to control the parameter while the animator is running. This variable input device frees the designer from the need for keying in individual data points, allowing him to concentrate on the actual evaluation of the display.

The animator will verify that at all times the display design conforms to the restrictions on size, shape, and color specified by the designer. Again, the display shape and size will be mapped to screen characteristics of the graphics terminal. The designer shall be able to magnify and demagnify the display design during the animation process so that he can more closely inspect particular aspects of his design.

2.4.3 Code Generator Requirements

The system will provide for the automatic programming of the target device. The code generator will combine the formal display description with the target specific code and translate them into the target device's instruction set. This program is the executable display description. When more than one device is available as a target, the code generator will allow the designer to specify the target by name so that the correct instruction set is used.

The code generator will also map the display design to the specific capabilities of a given target device. This process will include mapping the palette used in the display design into the palette of the target. Warning messages should be given to the designer if the number of colors used in the display design may not be shown on the particular target device. This mapping of colors needs to be defined so that the relationships between colors are preserved as well as possible.

Another mapping will be to match the size and shape of the display design to the size and shape of the target device's display viewing area. This mapping should be as close to one-to-one as possible so that a line 2 inches long on the graphics terminal will map to a line 2 inches long on the target device. If the display will not fit perfectly, the display design will be rescaled to fill the target's screen as closely as possible. This scaling will affect the sizes of all objects in the display equally. Warning messages should be provided in such situations.

Another requirement for the code generator subsystem is that it allow the programmer to specify which parameters named in the formal design description are to be matched to which external values gathered from sensors or other equipment. These external data values such as sensor data and other inputs will be gathered using a subroutine written by the maintenance programmer. The data gathering will happen in real time, with the target device's screen being updated at rates in the realm of 50 Hertz--or greater--in response.

The code generator will be able to produce executable code for a variety of specified target devices. The programmer will be able to add a new device at will. Code will be generated from the knowledge of two items: the instruction set of the given device and the name of a routine which sets the chromaticity and value of individual pixels. Such a routine would be written by a maintenance programmer for each different target device. If the target device provides powerful graphics functions, the compiler will be able to take advantage of these functions through more routines written by the programmer. Such advanced routines are not mandatory to automatically program the target, but they may allow more complicated display designs to be updated at high speeds.

Under certain situations a target device may not be supported by the code generator. This may occur when the target is new to the facility and the code generator support software does not yet know the instruction set of the new target. Another situation may occur when a display design is very complex and so the executable description must be highly optimal to achieve the desired update rates. In these cases, a programmer may perform the operation of the code generator by manually combining the formal design description with the target specific information.

2.5 Hardware Requirements

These are the requirements for the system hardware, including input devices, output devices, and the host computer hardware. These requirements are not detailed because many choices are implementation specific. In Section 3, we present further specifications for the hardware after examining the more important implementation issues. This reflects the decision to choose hardware only after the system is well defined. The capabilities of computer hardware must be considered when making implementation-specific decisions. These considerations must be postponed as long as possible so that they do not overly influence the usefulness of the entire system.

2.5.1 Input Device Requirements

An input device is any piece of hardware which allows the designer to express commands to the system. Example input devices include keyboards and graphics tablets. These input devices must be easy and natural to use. Also, a minimum number of input devices must be used; more than two or three invites confusion and fatigue by forcing the designer to switch often from one to another.

Three different types of input are needed for this system. The majority of the time spent with the system will be during the design phase, and during this phase much of the time will be spent placing such items as the ends of lines or centers of spheres. A device which moves a cursor on the screen of the graphics terminal is needed. The cursor may then be used to place endpoints and other positional information. Example input devices include the mouse, graphics tablet with pen or puck, joystick, thumb wheels, and light pen.

Another type of input will be for entering textual information. Such a device would allow the designer to enter names of display designs or numeric data. Examples of such a device include the keyboard and speech recognition systems.

The third type of input is the variable input device mentioned in the section on animator requirements. This device will be attached to a display input value so that as it moves, the value of the display input changes. Examples of such a device include a rotating or sliding potentiometer.

The above types of inputs overlap. The keyboard could be used to move the cursor on the screen. A mouse, joystick or light pen could be used to select a character from a list or to recognize characters handwritten by the designer. However, combining both inputs into one can lead to an awkward operator interface. To move the cursor from the lower left corner to the upper right using the keyboard may require many keystrokes and is not nearly as direct as repositioning a pen. Also, pointing to one character from a list of over one hundred means time-consuming scans of lists. As an alternative, the system could recognize characters hand-drawn using a mouse or light pen. However, this would require an often lengthy training session before a new user could begin to design. Hence, it would be preferable to provide both types of input separately, perhaps allowing each to overlap to a limited extent.

Positional information, such as that provided by a light pen, may be used to control the variable input, where moving the cursor on the screen is equivalent to twisting a dial. The difficulty with using the positional information in place of a separate device is that often the designer will need to control several display input values simultaneously. This could be difficult if only one controller were available. It would be better to have a separate device with several potentiometers all in the same package which is used only by the animator. Too many potentiometers will result in confusion as to which corresponds to which input value, so no more than six to eight need be provided.

2.5.2 Graphics Terminal Requirements

The graphics terminal will allow the designer to monitor the process of creating the display design and to observe the display as it will appear on the target display device. It must be able to match, if not exceed, the target device in all aspects of graphical image generation.

2.5.3 Hard-copy Printer Requirements

The designer needs to be able to make hard-copy printouts of the static images and the formal display description for evaluation by others and for archival storage.

2.5.4 Host Computer Requirements

As in any system, the host computer must provide adequate resources to support the application. More detailed requirements will be specified in Section 3.

2.5.5 Target Display Device Requirements

Very few requirements exist for the target display device since the purpose of the system is to supply whatever information the target device needs to draw the display design. The most important requirement is that the instruction set of the device be available to those who write the support software for the code generator as mentioned in Section 3. However, the general characteristics are herein described so that the reader will know for which target devices the system will produce output. The following paragraphs specify important aspects of the target display devices. They describe how images are made to appear on the screen and how the images are passed to the electronic hardware which transfers the image to the screen.

In most cases the target will be either flat panel (dot-matrix) or raster (cathode ray tube) devices. This means that images are shown on the screen as a large number of closely-spaced, colored pixels. This is opposed to vector devices which draw lines only. Vector devices are not normally used in applications which require images to change rapidly because they are designed for very high quality images, not for very high speed. However, the technology is changing and the system may one day need to produce output for such devices. No major changes will be needed in the system to support vector devices because all that would be necessary is vector instead of pixel definitions of how entities are to be drawn in the target specific code. Where a circle was before drawn using a large number of pixels, it would now become a large number of short, straight lines.

The other characteristic is that most target devices are "double-buffered." In this context, a buffer is a place in which an image may be stored while it is displayed on the screen. Double-buffering means that while

one frame is displayed on the screen from one buffer, the next frame is being drawn internally into a second buffer instead of directly on the screen. When the new frame is complete, the two buffers will be switched so that the second buffer is now displayed and the first buffer is redrawn with another frame. Thus the viewer does not see the images being drawn on the screen piece by piece, but is presented with the entire frame at once. If he were to see the screen blank out and then be redrawn, the images would appear to flicker on and off. This technique is not necessary if graphics may be generated at extremely high speeds or if the displays are extremely simple, but it is generally needed in the majority of situations.

2.6 System Usage Requirements

In this section we will show, in general terms, how the designer is to use the system. See Appendix B for an example session which shows in detail how a simple display would be created.

2.6.1 Overview

A display is usually designed according to the following steps. The designer first roughs out a sketch of the display design on paper. He then uses his sketch to draw the graphic images, adding the appropriate dynamic controls, into the system using the editor. He next invokes the animator to evaluate his design using the supplied test data. The preceding two steps are repeated until the design is satisfactory. Finally, the designer invokes the code generator subsystem to generate the executable display description for a specific target device. If the system does not contain the necessary data for the target device, a programmer may take the formal display description and perform this process manually.

The following is a general description of how the editor, animator, and compiler would be used. This section merely specifies the general actions of the designer; the actual details are very implementation dependent.

2.6.2 General Usage Requirements

The system will use menus, prompts, and on-line help to make it easy for the novice to learn. It may be assumed that the average novice has previous experience with computers (use of word processors and other similar tools) and a technical background. He should be able to learn enough commands in half an hour that he will be able to create and animate a simple display such as that shown in Appendix B. As mentioned above, ease of use is also mandatory so that the intermittent designer does not need to relearn the system every time a new display is designed.

2.6.3 Using the Editor

As in all other parts of the system, the editor will be menu driven. The designer will not need to remember commands, and on-line help will always be available. It must always be possible to abort a command in such a way that the designer does not lose any work except that directly associated with the command. The designer will work on only one segment (both its static graphics and its dynamic controls) at a time.

The designer will be able to use items from the library of modules, segments, icons, and user-defined entities in a display. A previously stored item will be retrieved by either its identifying name or by selecting a picture of the item from a menu. Since the library of items at a given site will usually become quite extensive, the editor must support the nesting of libraries within libraries so that, for example, all compass needle icons could be in one library with those used in airplanes in one sublibrary and those used in ships in another. This is similar to the nesting of directories within other directories as is available in most modern computers' operating environments.

It will be possible to display or hide the various other segments of a module while working on a given segment; displaying another segment can help in aligning centers of reference or entities, and hiding other segments can reduce the clutter on the screen. To further facilitate the careful positioning of objects, the editor will provide temporary grids and other positional cues. The animator will also provide this capability.

2.6.4 Using the Animator

After invoking the animator, the designer will specify the source of test data. He will also give the initial update rate for the entire screen as full speed, as a fixed number of updates each time a key is pressed (so that the designer may step through frame by frame), or as x updates every y seconds. He will also be able to specify the use of variable input devices to control specific parameters as defined above.

Whenever a special key is pressed, a menu will be displayed to allow the designer to continue, stop, start over, skip ahead, move back to repeat part of the display, change the update rate, run the display in reverse, or to respecify which parameters are controlled by variable input devices.

2.6.5 Using the Code Generator

The designer will enter the name of the target machine on which the display design will run. A programmer will need to create subprograms which define how the target devices will draw the display design, and a subprogram which gathers the data from the external environment to pass the parameters to the display subsystem. The code generator will gather all of these and output the desired code using the vendor supplied target instruction set. The only other interaction takes place when the designer requests the target device to

perform impossible tasks (such as using more colors than are available), causing the system to generate error messages to the designer.

This task may be performed manually by a programmer if necessary. Automatically creating the executable code for the target display device can sometimes result in slower update rates because automatic code generators do not generally produce totally optimal code. An experienced programmer can usually achieve faster update rates. For very complicated and time-critical display designs, the programmer may need to perform the task of the compiler subsystem manually. To do so, he must have access to the formal display description at the symbolic level.

2.7 System Expansibility Requirements

Any system which is used for a significant amount of time will need to be updated to provide new capabilities. This is why it was specified that the system be expansible. As the field of display design becomes more advanced, new requirements may be added.

As specified in the requirements above, the system supports the automated design and programming of electronic displays. The system does not automatically design or improve the displays, but only helps the designer by providing more powerful tools than a sketch pad and pen. After the field of artificial intelligence becomes more advanced and after the science of designing optimal displays matures, this system may improve toward suggesting improvements in the display to the designer.

Such capabilities lie within the foreseeable future. An expert system could incorporate the knowledge of experienced display designers to recognize poor designs. It could recognize objects that are positioned too closely, information that changes too rapidly, and use of too many colors.

Some displays may provide optimal transfer of information to the display user but may be too complicated to be updated in real time. Another expert system might be used to help guide the reorganization of segments in the display so that higher frame speeds may be achieved. This expert system might recognize nonmoving images and reorder transformations.

Later it may even be possible to have a descendent of this system design the display with minimal human intervention. The system would prompt the designer for what inputs are to be provided and what information is to be displayed and then produce the final product based on previous display designs. An evaluator would then suggest modifications to the system. These modifications would be incorporated in the current display and also added to a knowledge base so that the system 'learns' how to make better displays.

3 System Implementation

The requirements presented in Section 2 define an idealized system to automate the programming of real-time electronic displays. These requirements were determined to be necessary for the creation of displays, without regard to whether they were feasible using today's technology. While many of the requirements are supported by current technology, some require a development effort in which the cost exceeds the return in savings. Furthermore, some of the requirements need technological breakthroughs before they can be implemented. While it is likely that the breakthroughs will happen, the system is needed as soon as possible and a delay is undesirable. A system fulfilling most requirements today is preferable over one fulfilling all requirements in 10 years.

In this section, we present a possible implementation of the system containing a large subset of the requirements and maintaining usefulness. The primary goal is to show herein that a useful implementation of the system is feasible at the present time. A secondary goal is to define an upper boundary on the amount of developmental effort which must take place by rejecting those approaches which are more costly than others. We recognize that this implementation is not the only or perhaps even the best one.

This section contains the following five subsections:

- o The Problems of Implementation,
- o Data Implementation,
- o Process Implementation,
- o Hardware Implementation, and
- o A Brief Description of System Implementation.

The first subsection, The Problems of Implementation, details the problems which would be encountered by system developers and suggests solutions to those problems. The Data Implementation section describes the implementation of the various data. Process Implementation shows how the processes which create the data may be implemented. Hardware Implementation contains suggestions on the selection of hardware which will support the above process and data implementations. Finally, A Brief Description of System Implementation summarizes our suggestions for system implementation.

3.1 The Problems of Implementation

First we will explain why some of the above requirements are too difficult to implement by describing the current technology and its shortcomings. Then we will show how these obstacles may be overcome using as much off-the-shelf technology as feasible. This section is divided into the subsections:

- o Current Technology and Its Limitations, and
- o Circumventing These Limitations

3.1.1 Current Technology and Its Limitations

Graphic displays present unique difficulties to the engineer because of the large number of operations which must be performed within a short period of time. For instance, to draw a cube the computer must first calculate the positions of the end points of each edge. These positions are used to determine which faces are hidden from the viewer. The viewable end points must be translated, rotated, mirrored, and scaled from the world coordinate system in which they are defined into the screen coordinates. This results in the definition of a set of polygons which must be displayed. These rectangular polygons are then intersected with the clipping boundaries to create an irregular outline of the face which will fit within the prescribed limits. Finally, the outlines must be filled in with the appropriate color. Thus the image of the cube is presented to the viewer.

Each step requires multiplication of matrices and decision making, and the large number of multiplications required combined with the pixel-by-pixel nature of the final image would have a significant impact in the computation time required to produce each new display frame. Multi-million dollar computers exist for which the large number of calculations pose no problem, but such computers would produce very little payback. As the power-to-price ratio increases, many of the operations will become more feasible at an acceptable cost. This is why we differentiate between the ideal and the practical requirements. The technology is expanding too rapidly to allow considerations of feasibility to drive the ideal system requirements.

Due to the limitations of the current technology, intersecting the face of the cube with the clipping boundaries is one of the more time consuming of the above tasks. This is because every line in the image must be intersected with the clipping limits. Clipping at the abstract level is the process of finding the intersection point or points between an equation and the boundary and then redefining the clipped equation with the intersection points as the new ends of the lines. Clipping against arbitrary boundaries is not implemented by current technology because of the complexity of finding the point of intersection between two general equations. For instance, intersecting two cubic-spline surfaces would require advanced numerical techniques similar to those involved in finding zeros of equations. Such intersections usually involve making an initial guess and refining that guess to the desired precision. The refinement can require tens or hundreds of iterations to find one point, let alone the hundreds of points needed by the system.

Efficient algorithms exist to perform this task, but they apply only to boundaries which are rectangular and parallel to the coordinate system. These algorithms compare end points of straight lines with the coordinates of a rectangle defined from minimum to maximum x, y, and z values. These simple comparisons allow a large number of lines to be clipped rapidly.

Another operation which is time consuming is that of determining hidden surfaces in three-dimensional scenes. One technique is to store the apparent distance of each pixel along with its color. When two pixels collide, such as when a tree is behind a rock, the apparent distances of each are compared and the further pixel is thrown away. This is very simple, but many screens have a resolution of over one thousand by one thousand pixels. Each frame on such a terminal could mean the storage and comparison of well over a million distances.

However, methods do exist which quickly determine which faces are hidden in three-dimensional shapes such as a cube or a pyramid. These methods are based upon determining if the vector perpendicular to a particular face is pointed away or toward the viewer. This technique will remove totally hidden surfaces such as the back side of a house, but does not apply to partially hidden surfaces such as a tree in front of a barn. Partially hidden surfaces may be shown in two ways: by using the apparent distance technique defined above or by simply drawing the hidden surface in full and then drawing the other surface on top of the hidden one. The latter method of drawing objects in the background first will always work, but is not necessarily feasible because of the increased complexity in the code. The code is much simpler when objects in an image can always be drawn in the same order, and rearranging the objects would be time consuming for complicated displays. This simplicity in code leads to much higher update rates.

Transparency could be implemented using a similar technique where the color of the transparent (or partially transparent) item is given a unique value. Collisions result in a blending of the colors instead of replacing the old with the new. However, the required calculations would be even more extensive since the old color must be retrieved before the new can be specified and since this would happen on a pixel-by-pixel basis. Another technique to show transparency is to define a sieve so that only parts of the background show through the transparent object, giving the effect of looking through a screen door. Many commercial graphics terminals support overlaying such sieves in the hardware. This technique does not allow a full range because only a certain set of sieve patterns are effective; once the holes become too far apart, the eye separates the two colors. Also, using the sieve technique to show transparency would not produce a realistic image when viewing through multiple objects which are partially transparent.

Smoothly shaded surfaces pose yet more problems. Many algorithms exist today which break a curve into discrete panels and then interpolate the colors across the surfaces. These algorithms, due to their pixel-by-pixel nature and the complexities of interpolation, are also too slow to allow images to be displayed at frame update rates that are acceptable. The mapping of textured patterns to surfaces is a generalized form of continuous shading and is thus even more beyond current capabilities.

Finally, showing the diffuseness of an object's surface would require tracing a set of rays from each light source to the viewer, following each as it reflects from object to object to determine its final apparent color. Such ray-tracing algorithms could also facilitate showing transparency, smoothly shading objects, and suppressing hidden surfaces. However, the algorithms are

too slow for use in real-time environments, even when a supercomputer is used, due to the probabilistic nature of diffuseness. Many of the traced rays never reach the viewer because they bounce off the object away from the viewer, so tens of thousands of rays must be traced to create accurate images.

The above processes are difficult to perform at high speeds because of the limitations of both the target devices and the graphics terminals. However, the graphics terminals usually do not have the same capabilities as the target devices because the latter are often developed especially for a particular application to take advantage of state-of-the-art--and consequently more expensive--technology. This means that the graphics terminals will have difficulty animating some display designs that may be perfectly capable of being executed on the target devices, especially in the area of the speed at which the frames are updated.

3.1.2 Circumventing These Limitations

Two approaches are suggested below to circumvent the above limitations on the graphics terminals. One is to use hardware to implement those operations which involve large numbers of calculations. However, not all operations can be implemented in hardware. So the other method of circumventing a limitation is to identify which requirements are not absolutely necessary at the present time and postpone their implementation until they are supported by technology.

3.1.2.1 Use of Hardware

The algorithms mentioned above are too complicated and therefore time consuming to allow them to be implemented in software for real-time environments using the current technology. However, custom computer chips that allow these algorithms to be calculated in the hardware of many graphics terminals provide reasonable performance at a cost-effective price. Many terminals have hardware which perform the scaling, mirroring, rotating, translating, hidden surface removal, and simplified clipping operations. Also, many have hardware which will fill polygons and depth-cue lines. Furthermore, many support the "double-buffering" mentioned in Section 2.

Many graphics terminals also improve screen update rates by storing the graphic representations of the objects which comprise an image in "display lists". A display list is simply a list of high-level descriptions of the parts of a graphic image. When a particular frame is to be drawn, one command sent to the graphics processor will cause that processor to draw the frame on the screen.

Even using the above techniques, many of the other controls and attributes specified in the requirements section are not supported in the currently available hardware or are too slow to be of practical use. The list of unimplemented requirements includes clipping against nonrectangular shapes, diffuseness, and full transparency ranges. Smoothly shaded imaging is available, but the hardware is still too slow to support update rates which would permit the user to evaluate the display properly.

3.1.2.2 Reduction of Requirements

The other option is to identify which requirements may be deferred until a later time. Given the characteristics of target display devices on which display designs are currently executed, some of the above capabilities will not be used in the near future. Of those which could be used, some may be left out initially in the interest of obtaining the system as soon as possible.

Some of the more advanced attributes and controls specified in Section 2 of this report may be left out. Section 2 implies that there are no limits on the use of different colors in display designs. However, unlimited availability of different shades and hues is only necessary for total realism. For the current display designs, a minimum of 256 colors (where color refers to the combination of hue, saturation, and gray level) would be adequate. Of these 256, at least 32 hue and saturation combinations with at least 8 gray levels should be provided. This represents the bare minimum which would be necessary--the provision of a more complete spectrum is encouraged.

Clipping along rectangles which are parallel to the screen coordinate axes is mandatory. It is needed by any display which might extend beyond the screen. It is also needed to allow one instrument to partially overlap another or to allow displays of three-dimensional terrains to occupy the screen with displays of instruments. However, circles and more general shapes are not as vital as the rectangular limits.

Transparency can be implemented with only four or five different levels provided. Though the eye can differentiate between more, such a subset would allow transmission to the viewer of hidden objects. This represents the true need for various levels of transparency. More levels reflect the desirability of greater realism, but such realism is not absolutely necessary for the short term.

Diffuseness also is not absolutely needed for the short term. Diffuseness promotes realism in displays as objects can be recognized more easily with this information; but, again, such realism is nonessential for the short term.

Finally, it probably will not be possible to draw displays at the same rate in a commercial graphics terminal as in the state-of-the-art target displays. As a minimum, however, the graphics terminal should be able to achieve at least 25 frames per second when the target can draw 40 to 50 per second. This minimum allows evaluation to be useful even though the designer may perceive problems which will not appear in the final product because images do not move across the screen as smoothly.

All requirements which were not specifically addressed above should be implemented in full. This especially applies to those which provide ease of use. The success of any system is very dependent on the user interface. Making a system difficult to use by ignoring readily available technologies and techniques in order to save on development costs is generally a poor practice. Allowing the user to voice commands instead of typing them would require the solution of many problems facing those developing speech recognition systems.

However, the use of menus or graphics tablets does not require any technological breakthroughs. These available techniques for providing an effective user interface must be used to their full advantage.

3.1.3 System Implementation Partitioning

A system which implements the above reduced requirements could be custom built from scratch, but such a strategy would lead to a longer development time and higher system cost. A good implementation of any major system is one which builds upon as much previous work as possible. The desirable nature of graphics in many applications has lead to much development, and many of the components of this system are commercially available. Such components may not fully support the minimum set of requirements, but extensions may be made so that they do.

In the rest of Section 3, we present a more detailed system partitioning which takes advantage of the current technology, maximizing strengths and minimizing shortcomings.

3.2 Data Implementation

This section describes the implementation of the various data in the system. These include the display design, the formal display description, the executable display description, the library, the test data, and the target-specific code. The following sections parallel those in the section on Data Requirements.

3.2.1 Display Design Implementation

The display design is a concept. Questions of how to implement it are actually questions of how the system will support the hierarchy, attributes, controls, and other aspects of the display design. As mentioned in Section 3.1.2.2, some attributes and control information associated with display designs may not be supported by certain portions of the implemented system. These portions must be able to expand to include unimplemented attributes and controls as technology permits.

The only other aspect of the display design concept which needs to be addressed is how user-defined entities will be supported. They will probably be defined by computer code written in a high-level programming language. For example, if a view of the terrain is needed in a particular display, a programmer will write a subroutine which generates the scene on the display device given the relative position of the viewer. Since terrains are complex and since the amount of data needed to generate the scene would be extensive, the subroutine will probably need to access a large data base stored on such high-volume media as optical disks. Such a data base could be manipulated by the system, but this would be very inflexible. Thus user-defined entities will often need to be generated by computer code. The implementor of the system must give examples of how to create user-defined entities so that the maintenance programmer can create those needed for a particular application.

3.2.2 Formal Display Description Implementation

As the pivotal data in the system, the form of the formal display description can help or hinder the implementation of the remainder of the system. It must be in such a form that it may be easily created by the editor, easily used by the animator, and easily translated by the code generator into the instruction set of the target device.

There are several ways in which this formal description may be specified. One is the use of some special code specified by the implementors of the system. This is called the 'ad hoc' technique below. Another is to use a standard format such as the Initial Graphics Exchange Specification (IGES). A third method is to use a programming language such as Pascal or Ada(*). We will discuss below the advantages and disadvantages of each technique.

The ad hoc technique has the advantage of brevity. The special codes might describe a line from the point (3,23) to the point (10.2,0.4) as "1,3,23,10.2,0.4", and a circle centered on the point (-1.7,-223,5) with a radius of 0.04 units as "2,-1.7,-223,5,0.04", where "1" and "2" denote a line and a circle respectively. This can be very compact because no unnecessary information is provided. The minimum number of symbols needed to specify a circle is one denoting a circle, those specifying the location of its center, and one to specify its radius. However, this brevity complicates the programmer's task when manually writing the code for the target display because he must recognize the shapes from the symbols alone. If he cannot remember a particular code, he must look it up in a manual. This can be very time consuming. To remove this problem, a less compact but more descriptive format can be used.

The second technique, that of using a more standard format such as IGES, is essentially equivalent to the first except that it is already specified for the system implementor. This format can also be compact though it is normally not quite as compact as the ad hoc format since the standard format is designed for any application involving graphics. For example, a line entity in the IGES format is specified by eighty characters, many more than the eleven used in the above example. Where this technique loses compactness, it gains in allowing the system implementor to use previous work. However, as in the ad hoc technique, use of a standard format necessitates that the programmer learn which symbols represent which entities. This may be a more difficult task because the standard format may not support the display hierarchy, requiring the system implementor to improvise so that this information is transferred to the programmer.

The final technique is to use a standard high-level programming language. In this case, the description of the above line is an instruction such as `DRAW_LINE(3.23, 10.4)`. This is much more readable to the programmer because of its directness. However, it is less compact than the ad hoc technique.

* "Ada" is a registered trademark of the U.S. Department of Defense (AJPO).

Another advantage of using a standard high-level programming language is that the animator and code generator subsystems are reduced in complexity. This will be shown below in the sections describing the implementation of each.

Each of the above techniques can be made to work in the implemented system. However, the use of a programming language is the most promising. While the others can be made to be readable, the programming language most readily enforces readability. Also, it lends itself to using commercially available software as shown below. The consideration of compacting the information to save space is not as important as one might suppose. The increase in space is offset by a decrease in system development costs because of the use of nondevelopmental components. Since mass storage of information is readily available at reasonable costs, considerations of ease of development and usage are more important. Finally, a programming language supports specification of the display design at multiple levels.

Once the decision is made to use an existing programming language, the next step is to determine which language to use. A large number of likely candidates exist such as Pascal, Ada, and FORTRAN. Essentially, any language would be capable of formally specifying display descriptions.

Ada presents itself as the best choice for two primary reasons. First, the Department of Defense (DoD) has mandated that Ada be used for all software written for use in embedded computer systems. If any other language is to be used, special permission must be granted by the DoD. Second, it incorporates many advanced features which facilitate the description of display designs.

As a result of the first reason, programmers must be familiar with the language if they plan to work on DoD software projects, so very little training will be necessary before they are able to understand the formal display description. Furthermore, the language will have extensive support within the defense-related industries, so many of the pieces of the system which will work with the formal description will already be developed.

Ada also lends itself to the specification of the formal description because it incorporates many important capabilities. It was specifically designed for embedded microprocessor environments such as the target display devices likely to be used with this system. This means that low-level control of the microprocessor is readily available. Its multitasking capabilities allow a clean separation of data gathering and screen updating in the formal description. But most importantly, its packaging concept greatly reduces the complexity of specifying the display design at multiple levels of detail.

'Packages' are collections of subroutines which perform related sequences of actions. Each package specifies how other subroutines are to invoke those within the package and how each subroutine within the package is to be actually performed. Thus the package concept separates the interface from the implementation, so the implementation of a sequence of actions can be changed without affecting the rest of the program. For instance, a package of math routines might provide a function which evaluates the arccosine of a number using a series expansion. This might then be used in another function which calculates the angle between two vectors. If at some later date a polynomial expansion is determined to be more efficient for evaluating arccosines, the new method could be implemented without affecting the angle evaluation function in any way.

The packaging concept is useful in this application because of its ability to express the layering of the formal description. To draw a cube, one would invoke a DRAW_LINE subroutine which might in turn invoke a DRAW_POINT subroutine. Hence, the display is defined at the level of a cube, a sequence of lines, and a set of pixels. Furthermore, if a particular target device already knows how to draw a line, the implementation of the DRAW_LINE subroutine will be an execution of the appropriate instruction instead of a series of calls to the DRAW_POINT subroutine, without changing the drawing of a cube. Each target device would have a set of packages which allow the system to build the display design on that target's screen. Ada thus becomes a "virtual interface", i.e., an interface which allows the formal description to be target independent.

3.2.3 Executable Display Description Implementation

This will be a file of machine-level instructions for the target display created by the code generator subsystem. This file may be transferred to the target display device in any one of several ways, depending on the installation and the requirements of the particular device. One method is to use software which transfers files from the design system to the target device over a direct cable link. Another is to load the executable code into a programmable read-only memory (PROM) device which would then be placed in the target display device. These are two of the standard techniques used in the industry when programming embedded microprocessor devices. There are many other methods, and the choice of which to use will depend heavily on the particular situation.

3.2.4 Library Implementation

The format of the library will be determined by the needs of the editor subsystem. It will be specified by the implementor of the system.

3.2.5 Test Data Implementation

The format of the test data is determined by the needs of the animator. It will be specified by the implementor of the system.

Test data may be created in any one of several ways: by passing a magnetic tape recording of an actual run of the various subsystems through a program which filters out all unnecessary data, by gathering the data from the various subsystems in a simulation environment, by having a host computer simulate the generation of test data from calculations, or by the user interactively entering new values for data points in the proper sequence using a special program.

These various techniques allow the user to quickly generate data to test a particular design under a variety of conditions: when the test data must be very realistic, when he must test 'impossible' conditions, and when no actual equipment yet exists from which to gather the data. Such flexibility will improve both the accuracy and the speed of the evaluation process. The system needs to be open ended to allow a particular facility to devise its own techniques in data generation.

3.2.6 Target-Specific Code Implementation

The target-specific code will be a set of Ada packages which state how each target draws the various entities such as points, lines, and arcs as mentioned in Section 3.2.2 above. Each of these packages will be written by a maintenance programmer. Alternatively, the vendor of the target display device can write these packages. The key is that once the problems are solved for a particular device, they remain solved. The target-specific code also contains data-gathering subroutines for use in the final environment. These are also created by a maintenance programmer.

An important aspect of separating the formal description from the target-specific entity-drawing routines is that this allows the programmer to use more than one approach toward actually drawing the entities. Routines which draw lines, for instance, can either draw very accurate lines slowly or less accurate lines quickly. On raster graphics display devices, quickly drawn lines have a "stair-step" appearance because of the discreteness of the pixels. "Dithering" algorithms exist which draw better lines by controlling the intensities of pixels around the lines as well as those directly on the line, varying the intensity of a pixel with its distance from the true line. Dithering algorithms are slower but produce higher quality images. The maintenance programmer, when providing the routine which draws a line, can fit the tradeoff between accuracy and speed to the needs of a particular application.

The implementor of the system needs to provide examples of the target-specific code so that the maintenance programmer will know how to add a new target device. The examples also need to show different ways to draw such entities as lines and circles with tradeoffs between speed and accuracy. Furthermore, the implementor must clearly document the interface between the formal display description and the target-specific code so that the programmer can work at all levels of complexity from the pixel level up to the module level. These interfaces are also needed when the programmer defines entities for use by the system. Finally, examples of data-gathering routines need to be provided to further assist the maintenance programmer.

3.3 Process Implementation

This section describes how each of the processes involved in producing the above data (the editor, animator, and code generator), may be broken into subprocesses to make use of nondevelopmental software. Software is available which fills part of the need but not all, and the following sections identify

which parts can be filled with existing software and which cannot. A more detailed partitioning is shown by a figure for each process to show the use of nondevelopmental software. These figures use the same symbology as that in Figure 2-2.

3.3.1 Editor Implementation

The editor subsystem may be divided into three distinct pieces: a graphics editor, a dynamics editor, and a translator. This is shown in Figure 3-1. The graphics editor handles the definition of the shape of objects with their attributes while the dynamics editor handles the specification of attributes and dynamic controls at the segment level. This division allows the use of commercially available graphics editors. Furthermore, the output of the editor subsystem would not go directly to the formal display description, but instead goes to its own data base. This allows the editor to use a more efficient data-base format than that in the formal display description, and it also allows the use of nondevelopmental graphics editors. While this data base is formal, it is not readable by a programmer. Thus, a translator is needed to produce the formal display description.

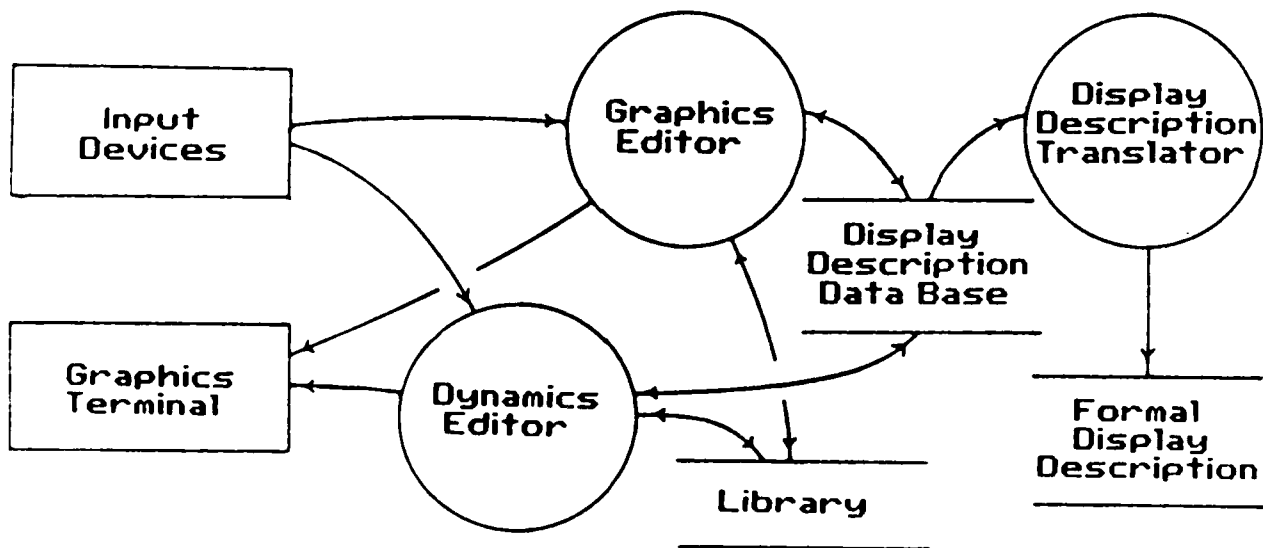


Figure 3-1 The Editor Subsystem Partitioning

3.3.1.1 The Graphics Editor

The graphics editor handles defining the shapes of objects and their attributes. Several types of editors which manipulate graphic images exist. These include programs which allow the user to "paint" on the screen and computer-aided design programs. None of these allow specification of dynamic

information to the full extent needed by the designer, hence a separate dynamics editor is needed to provide such capabilities.

Graphic painting programs are widely available for personal computers. These allow the user to "paint" an image on the screen by changing the colors of the individual pixels in a manner which imitates the artist's paintbrush on canvas. What these programs gain in simplicity is lost in power. These programs work at the pixel level and not at the entity level. The initial drawing of images is very easy, however, moving an entity on the screen can involve writing over the old with the color of the background and then redrawing the new in the correct position. These programs store the drawings as a copy of the bit map of the screen, where a bit map stores the color of each pixel and nothing more. This storage technique leads to simplicity because each command only effects the colors of individual pixels. More complicated techniques store more abstract information and require more bookkeeping to be done in each change to the drawing. Bit-mapped storage methods lead to difficulties in transmitting the display design to the programmer at many levels.

The major difficulty is in determining what entities make up a display design from only the knowledge of colored pixels. It is very difficult to determine from a picture of two connected squares if the image is actually one of a cube. Line drawing analysis for image recognition is an important issue in the field of artificial intelligence, and the associated problems are only partially solved. Other disadvantages include the fact that fine adjustments are very tedious and difficult, and that personal computers rarely support very high resolution or large numbers of colors because they are designed to be affordable for individuals, not to represent the state of the art in technology.

The second possibility for a graphics editor is to use a computer-aided design (CAD) package. In the engineering field, these packages are used to design and model everything from electronic circuits to car bodies. While many CAD packages also do extensive analysis of the models, this system will only utilize the ability to create and modify graphics. The fundamental characteristic of any CAD package is that it is a powerful graphics editor. The implementors of CAD packages have already solved the problem of drawing and redrawing pictures quickly and easily.

CAD packages deal with entities at the entity level. An entity is manipulated as an entity as long as it exists, not just until its initial position and size has been chosen. Storage of symbolic, abstract information is the major advantage of CAD packages. Internally, a circle is represented by some tag which denotes the entity combined with its center and radius. This is very close to the ad hoc description of the formal display description as described in Section 3.2.2. Thus, when the designer wishes to move the circle, he may pick up the circle and drag it across the screen to its new position, without manually erasing the old image. CAD systems extend such operations from the level of the entity to the level of a group of entities so that moving a representation of a vehicle, for instance, does not mean whitening out the old image line by line, curve by curve, just to redraw it somewhere else. Since this knowledge of entities and groups of entities is stored in the data base, recognition of these in the display design is very simple. The one disadvantage of the CAD packages is that they are expensive because of the amount of effort needed to develop a graphics editor at the entity level.

Given the advantages of using a CAD package, the painting programs available on personal computers are not adequate. Use of a commercially available CAD package will drastically reduce development costs by allowing its designers to solve the difficulties of letting the user easily draw graphic images.

The CAD package chosen must have several important characteristics. First, its internal data base must be accessible. This means that the formats of any files must be readily available to the implementors of the system. Second, it must meet the above requirements for ease of use or allow improvements to the user interface. Third, it must support the display design requirements as specified in Section 2. Fourth, it must allow the system implementor to make minor modifications and extensions which enable the package to be incorporated in the system. Extensions need to be added to provide for user-defined entities, to limit the available colors, to specify the target display size, and to allow the editing of dynamic controls. These extensions are described more fully in the following paragraphs.

Some user-defined entities need to be specified by software because of their complexity. For example, a three-dimensional map of the terrain might be defined by a complex data base. This data base then needs to be transformed in an efficient manner into the images placed on the screen. Some way to interface these user-defined entities with the CAD package needs to be developed.

Limiting available colors and specification of target display size allows the user to impose constraints on the display designs so that they are optimal for a particular target. Such constraints need to be removable because of the necessity for flexibility in the display designs. The user must be able to state on which target the display design is to be implemented by giving the name of the particular target device. The CAD package needs to be extended so that those constraints are enforced in display designs.

At least three ways need to be provided to specify the chromaticity aspect of color. One is by entering a name. The list of possible colors includes white, black, red, blue, green, and others added by the designer as needed. This will make it simple to enter commonly used colors. Another technique is to pictorially select a color from such a chart so that the full spectrum may be used. A third is to enter indices into a pre-defined scale or chart, such as the 1931 CIE (Commission International de l'Eclairage, or International Commission on Illumination) (x,y) Chromaticity Diagram. Likewise, gray level needs to be specified either by name--such as bright, normal, or dim--or by an index into a scale.

Finally, specification of the dynamics will require a custom editor. Several CAD packages provide for small, repetitive movements in images, but these movements are not general enough for this application and the specification is too difficult for all but the more advanced users. Hence a separate dynamics editor is needed and the CAD package must be extended to allow smooth transitions from one part of the editor subsystem to another so that user is not aware of these transitions.

3.3.1.2 The Dynamics Editor

The dynamics editor supports the specification of changes in attributes and positions of the segments in the display design. There are several ways in which this can be specified. One is to have the designer create Ada source code. Another is to trace the movements made by the designer and then interpret these movements into commands. A third is to use special icons to specify the paths for various parts of the segment. The last is to have the designer create controls using a specially developed language.

One possible way to specify the dynamics of a display is to have the user write Ada source code which invokes the appropriate commands in the appropriate sequence. Thus he might write a section of code which says "move the segment to the right three inches and then draw the 'ship module'". While allowing full control of the display, this increases the time novices must spend before being able to design even the simplest displays. Learning to specify motion might take weeks or months instead of minutes. While this technique allows entities and segments to be incorporated into display designs easily, it is no different than the current method of creating displays in that there is no intermediate step providing communication between the designer and the programmer.

Having the system track the designer's movements is at the opposite end of the spectrum. Using this technique, the designer specifies that a needle (and the segment in which it is drawn) is to rotate by moving the cursor in a circular motion. While simple, this method has several disadvantages. One is that it would be difficult to control the motion precisely. Another is that making minor modifications to the motion would be very difficult--the user would need to re-enter the entire movement to make any changes.

This latter technique can be improved by using the graphics editor to specify the motion path with special icons. A set of points on the segment, one for each dimension, are chosen as key points. For each point, a path is drawn from a starting position to an ending position. An expression is specified (such as $[(\text{TEMPERATURE} - 4) * 5]$) which governs the motion. Finally, values are given which correspond to the starting and ending points of the path.

The advantage of this technique is in its simplicity and preciseness. The disadvantage is that it does not allow full control of the sequence of transformations. It is very difficult to specify that a motion is to take place only if certain conditions are met. Furthermore, it may be necessary to limit the types of motion which may be specified to reduce the complexity of translating the paths into the formal description. Also, such a technique only applies to motion, not to chromaticity or other attribute changes.

The final method allows the user to specify all dynamics in a modifiable form without resorting to writing Ada programs. This is to write code in a special graphics control language using a custom, menu-driven editor. This technique is preferable to writing in Ada because the special language can be tailored to the needs of the system. Ada may be intimidating to the novice, but a graphics control language can closely parallel English so that it is more understandable. Also, Ada is not necessarily optimal for use by menu-driven editors because it was designed for general programming, whereas a

special language can be made to be optimal through the use of instruction formats which lend themselves to generation by templates.

The graphics control language will provide full control of display dynamics. It needs to incorporate the major control constructs of any programming language: sequential grouping of actions, conditional control of actions, and conditional looping. It also needs to incorporate the other major features such as variable manipulation and function invocation. Finally, it needs to support description of all of the dynamics which are supported by the system.

The best choice is to provide both of the above two concepts so that all capabilities are available. As the designer specifies a motion path for a segment, the corresponding graphics control constructs are generated automatically by the system. Then he uses the menu-driven editor to modify and embellish the dynamics, giving him full control. Hence both simplicity and full control is provided.

It is important that the formal display description be error free. By this we mean that nothing in the description should violate the rules that make it formal. For example, if two points are needed to specify a line, supplying only one point produces an error. This consideration is especially important when using an editor to create dynamic descriptions with a graphics control language. The editor must diagnose and help fix any mistakes of the sort described above.

3.3.1.3 The Translator

The output of the graphics and dynamics editors goes to a data base which is in a format recognized only by the two editors. This format will be dictated by the choice of a particular CAD package. While this editor data base will be formal, it will not be readable by a programmer. A translator will thus be necessary to create the formal display description.

This translator will be a further software addition to the editor subsystem. Its existence needs to be totally transparent to the user, and it will run concurrently with the graphic and dynamic editors, translating data as they become available. This minimizes the delay between finishing an editing session and animating the display design. Ideally, the user will be able to switch instantaneously from the editor subsystem to the animator subsystem so that he can create an idea and immediately try it out. Though we recognize that this may not be feasible given today's technology, effort must be made to reduce any delays.

3.3.2 Animator Implementation

The animator subsystem must transform the formal display description into moving images on the graphics terminal. To accomplish this, the animator subsystem may be divided into two distinct pieces: a compiler and a processor. This is shown in Figure 3-2. The animation compiler translates

the formal display description in whatever way is necessary into some format which will allow the processor to draw the images on the graphics terminal at optimum speed. The processor does the actual animation, combining the stored test data with those generated by the variable input devices as directed by the designer. The output of the animation processor goes to the same graphics terminal as used by the editor subsystem so that the designer does not have to switch from station to station. Likewise, the input devices used by the animator are the same as those used by the editor.

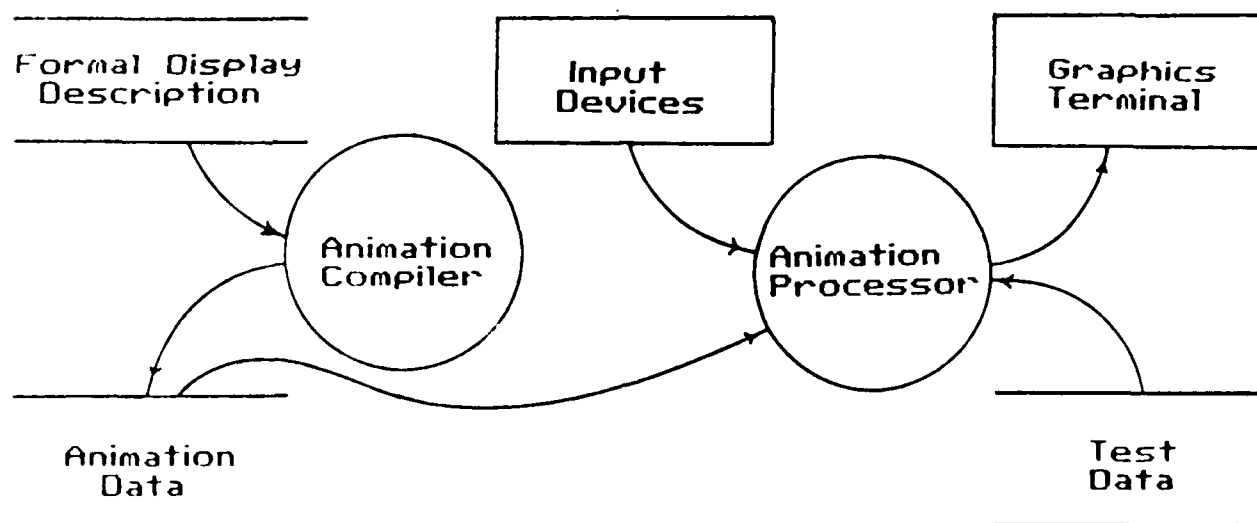


Figure 3-2 The Animator Subsystem Partitioning

3.3.2.1 The Animation Compiler

If Ada is used for the formal description of the display design, the bulk of this subprocess will be performed by an Ada compiler. The output of this animation compiler, i.e. the animation data, is a computer program which, when executed, will draw the images on the graphics terminal.

This compiler will be integrated with the translator in the editor subsystem so that as each module is defined, that module becomes available to the compiler. Thus the user will never need to wait for more than one module to be compiled before starting the evaluation of the display design.

3.3.2.2 The Animation Processor

This portion of the animator subsystem actually animates the display design. It is simplified by the use of Ada since it consists of the animation data, the test-data-gathering routines, and the input device handlers. After combining the above, the resulting program is executed. During execution,

test-data-gathering routines collect all of the data for each individual frame and provide it to the rest of the program. This closely parallels the method of drawing the images in the target display device.

The input device handlers allow the user to control the data gathering, specifying that the gatherer do such things as throw away a certain amount of the data (skip ahead) or that it reads the value of a particular input-data parameter with a variable input device instead of from the test-data file. By controlling the test data, the user may easily control the animation process.

Due to the limitations of hardware capabilities, it may be difficult to allow full control of certain forms of test data. Reversing the order of reading the test data or skipping backward through the data improves the evaluation phase by allowing the designer to quickly repeat a certain sequence of events. However, some forms of test-data storage do not allow such controls. For example, it is usually impossible to read a magnetic tape in reverse. Where attempting to implement such capabilities for a particular test-data source will seriously impact the time spent developing the remainder of the system, the capability should be left out if it will not seriously impact the evaluation process.

The methods used in generating data need to be very well documented so that a maintenance programmer may implement new techniques as needed.

An important point to remember in the design of this system is that it must be of general usefulness, not designed around any particular set of displays. This generality means that the graphics terminal's hardware must be very powerful to overcome the limitations of the software. Because of this generality, the Ada source code created by the translation from the editor's internal data base to the formal display description will not be able to take extensive advantage of special graphics tricks. The speed of the animator must come from a combination of a generally good translation and from very high performance hardware within the graphics terminal.

3.3.3 Code Generator Implementation

Finally, the code generator subsystem may be partitioned into a target-specific compiler and linker as shown in Figure 3-3. The compiler translates the formal display description into the instruction set of the target device. The display description code is then merged in the linker with the target-specific code, which contains calls to high-level graphics and mathematics routines on the target device if they exist, to produce the executable display description. The target device is specified by name, and this name is used by both processes seen in Figure 3-3 to generate the code for the correct target.

3.3.3.1 The Target-Specific Compiler

Again, the code generator subsystem will be simplified by the use of Ada in the formal display description. An Ada cross compiler may be used to create the display description code, where a cross compiler is a compiler

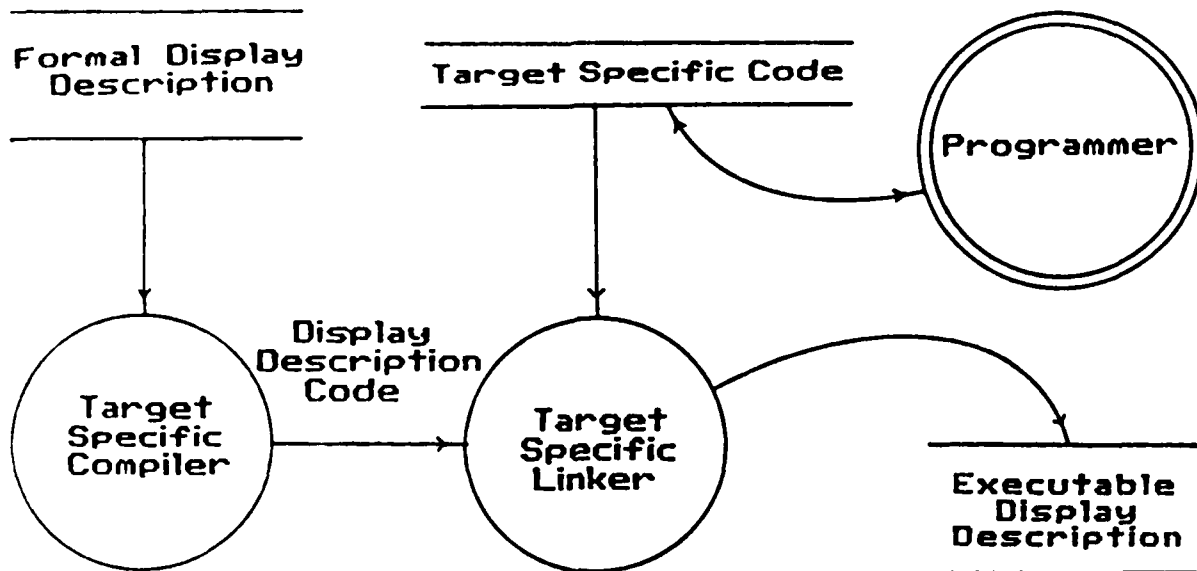


Figure 3-3 The Code Generator Subsystem Partitioning

which runs on one computer but generates machine code for another. Since the Department of Defense has mandated the use of Ada, Ada cross compilers should exist for any microprocessors used within the target display devices. A different Ada compiler will be used for each different type of target, and support software will execute the correct compiler based upon the name of the target device.

3.3.3.2 The Target-Specific Linker

Once compiled, the display description code will be linked (i.e., merged) with the target-specific code. This process is dependent upon the form of the output of the target-specific compiler, so a different one will be needed for each different compiler. Normally such a linker is available for each commercial compiler. Support software will control which linker is used in a particular case. In some instances, but not always, the link step will be performed by the software package that does the compilation.

3.4 Hardware Implementation

This section describes possible implementations for the input and output devices and the system computer. This section is purposefully last because of the need to have hardware choices driven by the data and process implementations.

3.4.1 Input Device Implementation

As mentioned in Section 2.5.1, the input devices must have two important characteristics: ease of use and familiarity to users. Another requirement is that those which will be used in the editor interface smoothly with the commercial computer-aided design (CAD) package. In this section, the devices which fill the needs most closely are presented.

Three types of inputs are needed for this system. One is for text and numeric data, the second is for positional data, and the third is for test data created by variable input devices during animation. For the text, the keyboard and speech recognition systems will be considered below. For the positional information, the mouse, graphics tablet and pen, graphics tablet and puck, light pen, joystick, thumb wheels, and the possibility of developing other devices will be considered. For the test data, different forms of potentiometers will be considered as well as the possibility of using the positioning devices.

Keyboards are the most widely used devices for inputting characters. Due to a long history of use in typewriters, keyboards are very familiar to most potential users. Voice input systems represent a strong alternative. They have the advantage of more rapid transfer of information to the system. The disadvantage is that they are still under research and practical speech-recognition systems are several years off. Hence the keyboard is the most likely device for the short term. However, the system should be implemented so that once the voice input becomes available, it may be integrated with a minimum amount of redesign.

As mentioned above, there are many ways to input positional information. The mouse is a small box which tracks its own motion through a ball or reflected light to move the cursor on the screen. If the mouse is moved without touching the desk top, the cursor does not move. The graphics tablet reads the position of a pen or puck to place the cursor. When the pen or puck is lifted, the cursor jumps to wherever it is put down again. The difference between the pen and puck is that the former looks like an ink pen while the latter is more similar to a mouse. The light pen has a light beam which is pointed directly at the screen of the graphics terminal so that no cursor is needed. The joystick controls the cursor by "steering" it about the screen, moving the cursor whatever direction the stick points. Thumb wheels have a separate wheel for vertical and horizontal cross-hair lines, and a thumb and finger are used to move each line.

The advantages for the various devices are as follows. The light pen is useful for pointing directly at an object on the screen. The mouse is the most familiar device because of its prevalent use with personal computers. The graphics tablet and thumb wheels allow for precise placement. Thumb wheels are particularly easy to switch to when entering a large amount of text, since they are often attached directly to the keyboard.

The disadvantages of each are as follows. Those with separate pens such as the light pen and graphics tablet with pen make it difficult to switch from the keyboard to pen and back. Graphics tablets take up a large amount of desk space. The light pen is tiring to hold for long periods because of its weight and the awkward position in which it must be held. The mouse can be difficult

to use when moving long distances across the screen because of limited desk space. The light pen and mouse are difficult to use when attempting fine control. The joystick can be slow because the cross hairs are "steered" instead of moved directly.

Since each of the above devices has its advantages and disadvantages, provision needs to be made for as many as possible. In particular, the mouse, graphics tablet and puck, and thumb wheels provide good control without too much frustration. Many CAD packages allow the user to make the decision, and the rest of the system should also allow such choices.

Alternatively, new devices might be developed. One example is a ball held in the air which operates as follows. Moving the ball up and down moves the image up and down on the screen, side-to-side motion moves the image side to side on the screen, back-and-forth motion zooms the image in and out, and twisting and rotating the ball twists and rotates 3-D images on the screen. This ball will be very good at positioning the image for viewing from all directions, but would be poor at specifying accurate points.

Another useful input device might be a digitizer which uses lasers to read the three-dimensional coordinates of an object. Such a box will make it simple to input accurate models of airplanes, vehicles, and the like. The system should be open ended enough that it allows such devices to be included at a later date.

As mentioned above, the various forms of potentiometers could be used to input test data during the animation phase. More than one vendor has a box which has several knobs to control several potentiometers. Each potentiometer could be treated as a separate source of test data to allow simultaneous control of several parameters in the display design. Some of these devices have small, six to eight character displays, which could be used to note which knob is tied to which parameter or to display the value itself. Since these devices usually interface only to the graphics terminal made by the same manufacturer and since their capabilities are not as significant as other considerations, undue emphasis should not be placed upon selection of a variable input device.

The positional input devices such as a mouse or light pen could be used to provide the functionality of the above potentiometer box. Techniques could be developed to allow a mouse to control several parameters. An up and down motion could change one parameter while a sideways motion would change another. Using such devices as the graphics tablet or light pen would minimize the amount of equipment at the designer's workstation at the expense of being able to control only a few parameters.

3.4.2 Output Device Implementation

In this section, we present possible devices which output human-viewable pieces of the display design.

3.4.2.1 Hard-copy Output Device Implementation

Several devices are available which output to transportable media. These include printers, video cassette recorders, and photocopiers.

Literally hundreds of printers are available with many different capabilities at many different prices. Characteristics which influence the choice of a printer include colors available, precision, speed, clarity, price and size of paper used. Very few printers provide as wide a palette as many of the commercially available graphics terminals and this could pose a problem in some situations. Many printers draw with such precision that full size printouts of cars are accurate to within fractions of a millimeter. Less accurate precision is needed by this system since the printouts will be used for general information transfer, not details. Likewise, the printouts do not need to be large as little need exists for wall-sized printouts of 12-inch square display designs.

Another technique for storage and transmission of display designs would be to use a video cassette recorder (VCR) to tape the images on the screen of the graphics terminal during the animation process. This normally works by attaching the VCR to special output leads from the terminal. This is useful when those who must evaluate a display design cannot come to the facility in which the graphics terminal is located. Due to the bandwidth of the tape, the taped images may not provide as high a resolution as the original. This technique provides the best way to store and transmit accurate representations of the display design concept.

Finally, some manufacturers produce devices that make a direct photocopy of the images on a graphics terminal without the use of a camera. These are preferable to paper printouts when very accurate renditions of the display are needed to show the effect of pixel resolution on the display.

Of the above three, provision for paper printouts is the most important. It is not necessary that the implemented system interface to either of the other two devices. If a video cassette recording is absolutely necessary at a particular facility, one can use a video camera to record the images directly off the screen. Likewise, the photographs can be created with the use of a good quality photographic camera.

The staff at each site in which the system is installed will need to determine which hard-copy outputs are needed for their application. The system implementor should provide a listing of the suitable printers and other hard-copy devices along with their capabilities and prices.

3.4.2.2 Graphics Terminal Implementation

There are two primary requirements for the graphics terminal: it must interface with the chosen computer-aided design package and it must be able to draw graphic images on the screen as quickly as the current target display devices. The following features are necessary on the graphics terminal in order to meet current capabilities and needs:

- o 1024 by 1024 pixel resolution, and
- o 256 colors (including at least 32 different chromaticity values at 8 gray levels).

The capabilities of the graphics terminal need to match as closely as possible, if not exceed, those of the currently available target display devices in screen update rate, and the ability to display images which meet the requirements. It should also provide a pixel resolution which is as fine or finer than the target devices.

As mentioned above, some graphics terminals implement various functions in hardware, and those terminals are recommended. In Appendix C we list and compare the capabilities of several terminals which provide such hardware.

3.4.3 Host Computer Implementation

As a final consideration, the host computer for the system must also have certain features. Given the large amounts of memory which will be needed to quickly update complex displays, at least a 32 bit machine supporting virtual memory will be required for the host computer. Virtual memory allows a program to be larger than the actual size of the memory, allocating sections of disk storage when the program is larger than physical memory. Many of the current host computers provide over eight million bytes of memory, but it cannot be guaranteed that this will be enough in every case. Since the amount of storage available on a disk is typically several orders of magnitude greater than physical memory, virtual memory goes a long way toward postponing problems of running out of memory. Thirty-two bit architecture is needed for the same reason: to provide rapid access to large amounts of data storage. In general, the host computer must provide as much memory as possible and execute programs fast.

Another requirement of the host computer is that it interface to the chosen graphics terminal and the chosen computer-aided design package. Furthermore, if Ada is chosen for the formal display description, an Ada compiler must be available for the host. Developing a custom Ada compiler will increase the price of the implemented system beyond reason.

Finally, it must be possible to interface the host computer to the needed output devices and to other computers. These interfaces must be fast, especially those to other computers, if those computers are to generate test data during the animation phase.

Possible criteria for the choice of a host computer include items other than the above necessities. Tape drives will be useful for making archival copies of formal display designs, target-specific code, etc. This will allow simple re-entry of the information in the case of the loss of data as when a disk becomes corrupt. Also, the ease of use of the operating system provided with the host must be considered.

3.5 A Brief Description of System Implementation

This section summarizes the possibilities for system implementation described in the preceding four sections. Two very important decisions drive the final system. One is the decision to use a commercial computer aided design (CAD) package to do the editing of the object shapes. The other is the decision to use Ada in the formal display description.

Since the editing of object shapes is provided by the CAD package, another editor must be developed to support the dynamic aspect of display designs. This editor should allow both graphical and textual specification of the dynamics. Also, since the CAD package supports its own data base, a translator is needed to produce the formal display description.

Once the formal description is created, a commercially available Ada compiler for the system computer will be used in the animator subsystem to provide motion to the display design. This compiler will generate the system-specific code which updates the screen of the graphics terminal at rates exceeding 25 frames per second unless the designer has slowed down the animation process. With the use of Ada, the only development involved in implementing the animator subsystem is in writing test-data-gathering subprograms and possibly developing software which performs minor modifications to the formal description to optimize it for the graphics terminal and host computer. In order to achieve 25 Hertz frame-update rates, the use of a powerful graphics terminal which provides transformations, clipping, and other algorithms in the hardware is needed.

Finally, after the display design is seen to be optimal on the graphics terminal, the code-generator subsystem automatically creates the program which will be downloaded into the target display device. The bulk of the processing is done by target-specific Ada compilers and linkers. Hence the development for this subsystem is limited to building an environment for the compilers and the programmer-written target-specific code. This subsystem uses the Ada packaging concept as a virtual interface between the generality of the formal display description and the particular features of the target device. The maintenance programmer will write data gathering routines for the particular application. He will also write and maintain the routines which draw entities on the screen of each target device at a particular facility.

4 Conclusions and Recommendations

In this report, the system requirements and top-level design have been specified for a system to support the design and automated programming of electronic displays. The requirements were determined for an ideal system not limited by current technology.

To support the creation of display designs, a hierarchy of levels was created with special capabilities and attributes attached to each level. The created display designs are expressed concretely in the formal display description. The formal description, when combined with test data, allows the designer to evaluate the display design. Once the display design is optimal, the automatically generated executable display description may be loaded into the target device for use in simulators, aircraft cockpits, command and control centers, or anywhere large volumes of data must be presented graphically at high speeds.

The system was partitioned to support the creation, test, and compilation of the display designs. This division of the system into an editor, animator, and code generator allows the use of commercially available hardware and software. Once the requirements for the editor were defined, it was seen that many of the problems associated with creating and modifying graphic images are solved in computer-aided design packages. This saves the development of tens of thousands of lines of computer software.

Similarly, once the requirements for the animator and code generator were specified, it was seen that the use of Ada to formally describe the display design meant that these processes could make use of Ada compilers. This again saves the development of large amounts of software.

In Section 3, it was noted that current technological limitations preclude the full implementation of all attributes and controls. However, the majority of them, including those which are the most important, can be filled at the present time. The modularity of the system design will enable additional capabilities to be added with minimal impact upon the initial design.

The system may be feasibly built at the present time for a reasonable cost by making extensive use of nondevelopmental hardware and software. The benefits to be gained by using this system are great, and it is recommended that a Phase II effort be funded to specify the detailed hardware and software requirements, create the complete system design, and implement the system as soon as possible.

APPENDIX A

GLOSSARY OF TECHNICAL TERMS

The terms in this glossary are defined as they are used in this report, and do not necessarily agree with their use outside of this context.

animator: The portion of the system that allows the designer to evaluate the display design by seeing it change on the screen in real time, simulating its appearance in the target environment.

attribute: An item of information which is associated with a graphical image and is distinct from the shape of the object. Examples of attributes include line texture and color.

bit map: A matrix containing the same number of elements as there are pixels on the display screen. Each element in the matrix is assigned a color, and the value of each element in the matrix can then be mapped to its corresponding pixel on the display screen.

computer-aided design (CAD) package: A software package that permits its user to create and modify graphic representations of two- and three-dimensional objects.

children: When used in the context of a hierarchical structure, such as segments nested within other segments, the "children" are those objects which are lower in the hierarchy than the object of reference. An example is the relationship between two people as seen on a family tree.

chromaticity: The portion of color not associated with gray level. Chromaticity is further broken down into hue and saturation.

clipping boundary: A closed two- or three-dimensional shape that encompasses an area or volume outside of which the entity or segment to which it applies will be clipped, i.e., not drawn.

code: When used as a noun, it refers to a computer program (either a human- or machine-readable program); when used as a verb, it refers to writing a computer program.

code generator: The portion of the system that transforms the formal display description into the executable display description.

color: The attribute of visual experience that can be described as having quantitatively specifiable dimensions of chromaticity and gray level. It does not include the portions of the visual experience dealing with extent (size, shape, etc.) or duration (movement, flicker, etc.).

coordinate system: A set of three orthonormal vectors defining a set of axes labelled x, y, and z which allow the unambiguous determination of the position of points in space. A local coordinate system for a segment is specified by placement of origin, orientation of axes, and the measurement system in use.

data base: A data file containing a usually large number of data records which are connected and can be cross referenced by various fields within the records.

designer: The person who creates the concept of a display and transforms it into a display design.

diffuseness: The attribute that specifies to what extent light from a light source will be specularly or diffusely reflected from a surface. It essentially defines the 'texture' of a surface, i.e., how smooth or rough it appears.

display: The time-varying visual mapping of the state of the environment onto the target device screen. The display is a sequence of static frames drawn by the target device based on input data values, and the impression of motion is created by the frames being displayed at a sufficiently great update rate.

display design: The abstract concept or idea describing how the visual mapping of the state of the environment will occur based on the values of and changes in the input data.

dynamic controls: The commands attached to a segment which control where on the screen a segment will be displayed as well as whether the segment will be displayed. These commands may also specify the format of displayed numeric and textual data. Examples include rotation about and translation along an axis.

editing: The process of creating and modifying data; in the context of this report, the display design.

editor: The portion of the system which transforms the commands representing the display designer's idea of the display into the display design.

entity: The smallest graphic object manipulated by the graphics editor. Examples include lines, circles, and cubes.

executable display description: The concrete representation of the display description as an executable computer program. It is also referred to as the executable description.

font style: The attribute which specifies the particular set of shapes to be used in displaying text and numerals. Examples of font styles include italics, gothic, etc.

formal display description: The concrete representation of the display description expressed in a high-level programming language. It is also referred to as the formal description.

frame: The static assemblage of images filling the screen at any one instant in time which represents the state of the environment based on the current input data values. The rapidly changing sequence of frames make up the display.

graphics terminal: The output device in the system which allows the designer to visually monitor the display design during the creation phase and to evaluate the display during the test phase.

gray level: The portion of color not associated with chromaticity. The gray level is the measure of the brightness or lightness of an object or a pixel.

hue: The portion of chromaticity that can be described by such words as 'red', 'green', or 'blue'. It specifies the wavelength(s) of light being emitted or reflected from a surface.

icon: A collection of entities stored in the editor library and manipulated as a unit.

image: The static visual representation of an object or group of objects displayed on the display screen.

input data: In the context of the target environment, the data passed to the target display device by external equipment such as sensors, other computers, etc. which are used to determine the appearance of the display.

input devices: A device used by the designer to give commands to the system. Examples include a keyboard and a mouse.

instruction set: The total set of microprocessor-specific commands that are available for use in a machine-language program written for a particular computer.

library: The data base used by the editor to store previously-defined portions of displays for later retrieval and reuse in other displays.

module: A collection of one or more segments which have been named and which specify the inputs to the display from the environment.

module invocation: The act of including a module within a segment by referencing it by name.

novice: A designer or other user of the system who is unfamiliar with the use of the system or computers in general.

package: A collection of subroutines that perform logically-related tasks. Packages are usually associated with the programming language Ada.

parent: When used in the context of a hierarchical structure, such as segments nested within other segments, the "parent" is that object which is higher in the hierarchy than the objects of reference. An example is the relationship between two people as seen on a family tree.

pixel: A contraction of 'picture element'. The pixel is the fundamental entity in graphics. It is the smallest resolvable area of a display screen, in which an average color value is determined and used to represent that portion of the scene being displayed. Pixels are arranged in a rectangular array to form the entire display screen.

priority: The attribute which specifies the importance attached to displaying a particular segment in the event that it and one or more other segments are to be displayed in the same or overlapping regions. The segment with the higher priority will be displayed over the other segment(s).

process: Something which transforms data into other data according to a specific set of rules. In the context of the system described by this report, the major processes are the editor, the animator, and the code generator.

programmer: In the context of the current method of creating displays, a person knowledgeable in the programming of microprocessors who takes a display designer's drawing and writes a computer program to implement it as a display. In the context of the system described in this report, either a person who manually converts the formal display description into the executable display description or a person who writes the general data-gathering and entity-drawing subprograms for a particular target display device.

real time: The ability to respond to changes in the environment within a suitably small period of time after the change.

resolution: The extent to which a display screen can accurately display an arbitrarily complex or small object, which is usually a function of the number and size of the pixels that make up the display screen.

saturation: The portion of chromaticity that specifies the proportion of pure chromatic (as opposed to achromatic -- white, gray, black) light in the total light emitted or reflected from a surface.

screen: The physical device to which the display is mapped. Examples of screens include cathode ray tubes, liquid crystal displays, etc.

segment: An element of the display design hierarchy. It is composed of entities, icons, subsegments, invocations of modules, attributes, and dynamic controls.

serial number: A number used to differentiate segments.

sibling: When used in the context of a hierarchical structure, such as segments nested within other segments, two objects which are children of the same parent are termed "siblings."

simulate: To execute the display design on the system in such a manner as to make the display behave the same as it would on the target display device, allowing the designer to evaluate the display design.

target display device: The microprocessor-controlled equipment located in the target environment which will visually map input data values representing the state of the environment onto a display screen in accordance with the executable display description executing on it.

target environment: The location where the target display device will be used to rapidly and efficiently convey information describing the environment to a display user. Examples of target environments include aircraft, ships, factories, etc.

test data: The data that simulates the data input to the target display device in the final environment. The test data are used by the animator portion of the system as the source of inputs used to drive the display design in the test phase of the display creation process.

transparency: The attribute that specifies to what extent the viewer will be able to see through an object to observe objects that are hidden behind it.

update rate: The rate at which a segment is redrawn using new input data values.

user: The display designer or any other person using the system.

user-defined entity: An entity which was not provided with the original system but was created (usually by a maintenance programmer) to represent a complex object or concept not readily represented by the entities provided by the system. An example would be a particular representation of a three-dimensional view of the terrain.

variable input device: A device such as a potentiometer or slide which allows a designer to incrementally change input data values during the evaluation of a display design.

viewer: The display user who visually gathers the information presented on the display screen by the target display device.

APPENDIX B

DETAILED OPERATIONAL SCENARIO: CREATING AN EXAMPLE DISPLAY

B.1 Introduction

This appendix describes the creation of an example display. This example will not be very detailed because much of the detail depends on the actual implementation of the system. We will assume the system is implemented along the lines as described in Section 3, but we will not assume any particular graphics terminal or computer-aided design package.

This example will not show every feature of the system. It will provide the reader with an overview on how many of the capabilities may be used to create a useful display design. The particular design shown below is possibly neither optimal nor desirable; it is for illustrative purposes only.

B.2 Sketching the Display

The first step is to decide what information is to be pictured. We will create here a display for depicting the levels of three fluids: fuel, oil, and coolant in a vehicle. We will display three items of information for each. The level of each in absolute numbers will be shown numerically. The level will also be shown as a fraction of the full capacity by the use of a dial. A scale will be used to show the rate of loss of the fluid. If the level drops below a certain threshold or the rate of loss climbs above another threshold, the entire display for that fluid will start to blink to draw attention to the problem. The final display will appear as in Figure B-1.

We will assume that the space allocated for the display is 2 inches by 4 inches so that each separate gauge has a space of 2 by 1-1/3 inches.

B.3 Determining the Display Parameters

After the basic concept is sketched out, the next step is to identify the inputs which will be needed. In this particular example, the inputs are the level and the rate of loss of each fluid. The latter information could be calculated internally within the display subsystem, but we will assume that it

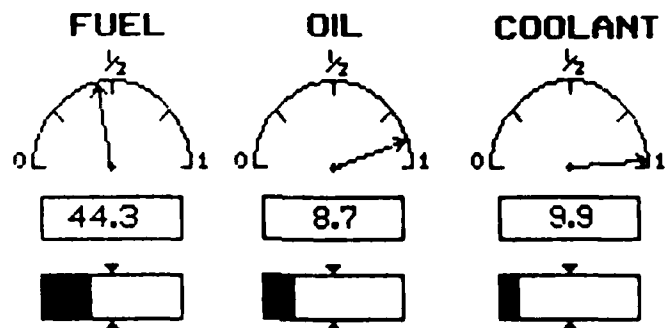


Figure B-1 The Fluid Status Display

is not for simplicity. We will name the fluid parameters 'FUEL_LEVEL', 'OIL_LEVEL', 'COOLANT_LEVEL', 'FUEL_LOSS', 'OIL_LOSS', and 'COOLANT_LOSS'. We will also assume that all levels are measured in gallons and that losses are measured in gallons per hour or per hundred hours.

B.4 Partitioning the Design

Next the display design should be partitioned into modules and segments. Since each set of gauges is to be alike, we will create a generic gauge module to show all three fluids. Then we will invoke this gauge module three times, once for each different fluid.

B.5 The Gauge Module

As mentioned in Section 2 of the report, every module has a defined set of inputs. Two of the inputs needed are the fluid level and rate of loss, named 'FLUID_LEVEL' and 'FLUID_LOSS', respectively. Also, the gauge module needs to know the various thresholds mentioned above. These will be named 'MAX_LEVEL', 'MIN_ACCEPTABLE_LEVEL', and 'MAX_ACCEPTABLE_LOSS'. These will be specified by the designer when he creates the module.

The gauge module can be partitioned further by determining the various dynamic elements. The needle will be rotated, so it should be in its own segment. The numeric value should have its own segment so that it may be displayed at a specified location on the screen. Finally, the scale will be implemented by rescaling a unit rectangle along the appropriate axis, thus it will also need to have its own segment.

In the next several sections, we will show how the gauge module can be designed using the system. Assume that all lines not otherwise specified are solid and white on a black background.

B.5.1 Segment 1.0

Segment 1.0 is at the top level of the module. It will contain two distinct items. One is the skeleton for the three gauges as shown in Figure B-2. The other is the set of commands controlling the blinking of the display.

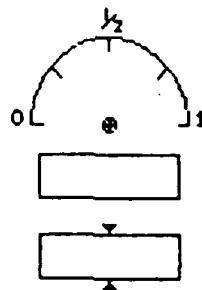


Figure B-2 Segment 1.0

To create the gauge outlines, the designer enters the graphics editor and performs the following sequence:

1. The designer places the origin as marked above by a cross in a small circle at the center of the figure. This special cross and circle will not actually be drawn in the final display.
2. He commands the system to draw a semi-circle, fixes the center at the origin, and specifies that the radius will be $7/16$ of an inch. This command might be in the form of pointing to a picture of an arc, pointing to a place for the origin, and entering the radius using the keyboard.
3. He placed the tick marks on the arc by drawing a line from the the center to the arc and erasing all but $1/16$ inch of the line.

4. Possibly using a menu, he commands the system to draw text and puts in the 0, 1/2, and 1 markings as shown.
5. He moves the cursor down 1/8 inch from the center of reference and to the left 3/8 inch, commands the system to draw a box, fixes the upper left corner, drags the cursor down to the desired position for the lower right corner, and fixes this as well so that the box is 3/4 inch long by 1/4 inch high.
6. He does a similar operation to create the lower box.
7. He places triangles at the top and bottom of the center of this second box to mark the maximum acceptable loss.

This completes the skeleton of the gauges at the Segment 1.0 level.

Now he must specify that the Segment 1.0 (and therefore all other segments) are to blink under certain conditions. He specifies that if FLUID_LEVEL is less than MIN_ACCEPTABLE_LEVEL or if the FLUID_LOSS is greater than MAX_ACCEPTABLE_LOSS, then the segment is to blink at the medium rate. This completes the definition of Segment 1.0.

B.5.2 Segment 1.1

The needle will be placed in this segment. First the designer specifies that Segment 1.0 is showing to provide a reference. Next the designer selects the appropriate needle icon from the library. Assume this icon is 1 inch long by 1/4 inch wide. The designer shrinks the needle so that it is just short of the '0' tick mark. This is the rest position of the needle. The designer then uses the dynamics editor to specify the rotation of the needle by an arc-like dynamics icon from the rest position to the other end of the scale. He then specifies that the rest position occurs when FLUID_LEVEL = 0 while the other end of the scale occurs when FLUID_LEVEL = MAX_LEVEL. The dynamics editor will determine that the rotation of Segment 1.1 is to be $[(\text{FLUID_LEVEL} / \text{MAX_LEVEL}) * -180]$ degrees.

B.5.3 Segment 1.2

Before defining Segment 1.2, the designer specifies that Segment 1.1 will not show to reduce the clutter on the screen while editing 1.2. The numeric value of FLUID_LEVEL will be displayed in the box in this level by specifying that it is to be displayed centered in a field with five spaces. No commas or signs are to be displayed. Two places after the decimal point are to be displayed. All of this is specified using the dynamics editor.

The position at which the data will be displayed will be the segment's origin. Therefore, the designer uses the dynamics editor to specify that the origin for Segment 1.2 is to be translated down and to the left so that it is just above the bottom line of the box and just to the right of the left line. The data will be centered in the five spaces to the right.

B.5.4 Segment 1.3

Finally, in Segment 1.3, the rate of loss of the fluid will be shown on the bar. The designer again requests that Segment 1.0 be shown along with Segment 1.3. He then positions the cursor just inside the line at the middle of the left-hand side of the bottom box, expanding the image as necessary to achieve accuracy. This point will become the origin for the segment.

Selecting the color yellow, he defines a box which is just inside the outer one and specifies that this box is to be filled in with a solid pattern. Next he enters the dynamics editor to specify that the yellow box is to be re-scaled along the x-axis. The rest position is when $\text{FLUID_LOSS} / \text{MAX_ACCEPTABLE_LOSS} = 2$ and the yellow box decreases to a thin yellow line when $\text{FLUID_LOSS} = 0$. The dynamics editor will then determine that Segment 1.3 is to be scaled along the x-axis by $\text{FLUID_LOSS} / (\text{MAX_ACCEPTABLE_LOSS} * 2)$ when this is between 0 and 1, and that if $\text{FLUID_LOSS} = 0$ a line is to be drawn. The yellow box will never go outside of the outer white box. Note that it will reach the halfway triangular markers when the loss is barely acceptable.

The dynamics editor also determines that the origin of this segment is to be constantly placed upon the left edge of the white box as defined above.

B.6 The Top Level Module

Once the above gauge module is defined, the next step is to create another module with inputs `FUEL_LEVEL`, `OIL_LEVEL`, `COOLANT_LEVEL`, `FUEL_LOSS`, `OIL_LOSS`, and `COOLANT_LOSS`.

B6.1 Segment 1.0

The origin will be the center of the 2-by-4-inch space. This segment will also be split into three subsegments, with one for each gauge. In the Segment 1.0, the designer will place the words 'Fuel', 'Oil', and 'Coolant' so that each is centered above the area where the gauges will be placed.

B.6.2 Segment 1.1

Segment 1.1 will be the left-most gauge which shows the fuel information. The designer specifies that the origin will be to the left of the Segment 1.0 origin by 1 and 1/8 inches. Using the dynamics editor, the designer then specifies that the Gauge module be invoked. The parameters are associated so that $\text{FLUID_LEVEL} = \text{FUEL_LEVEL}$, FLUID_LOSS , $\text{MAX_LEVEL} = 100.0$, $\text{MIN_ACCEPTABLE_LEVEL} = 1.0$, and $\text{MAX_ACCEPTABLE_LOSS} = 1.0$ gallons per hour.

B.6.3 Segment 1.2

Segment 1.2 will be the center gauge showing information about the oil level. It does not need transformation from its original position. In this instance, the Gauge module is invoked with `FLUID_LEVEL = OIL_LEVEL`, `FLUID_LOSS = OIL_LOSS`, `MAX_LEVEL = 10.0`, `MIN_ACCEPTABLE_LEVEL = 6.0`, and `MAX_ACCEPTABLE_LOSS = 0.1` gallons per one hundred hours.

B.6.4 Segment 1.3

Finally, Segment 1.3 will be the right-most gauge showing the coolant level and loss. Its origin will be 1 and 1/8 inches to the right of Segment 1.0's. The designer specifies that it also invokes the Gauge module with `FLUID_LEVEL = COOLANT_LEVEL`, `FLUID_LOSS = COOLANT_LOSS`, `MAX_LEVEL = 10.0`, `MIN_ACCEPTABLE_LEVEL = 8.0`, and `MAX_ACCEPTABLE_LOSS = 0.05` gallons per one hundred hours.

B.7 Conclusion

The above example shows how a designer can create a simple, two-dimensional display. This display shows the usage of nested modules and segments to a limited extent. It also shows the usage of the capabilities of the graphics and dynamics editors. It is hoped that this appendix leaves the reader with a firmer understanding of the use of an implemented system.

APPENDIX C

COMPUTER-AIDED DESIGN PACKAGE AND HARDWARE SURVEY RESULTS

C.1 Introduction

In Section 3 of the attached report, we present recommendations for partitioning the system into pieces which use nondevelopmental components to reduce the effort required for system implementation. This partitioning is based on the use of commercial computer hardware and computer-aided design packages. We performed a preliminary survey of the available technology to determine if products exist which fit the need. The purpose of this survey was not to find the absolutely best product, but to determine if our recommendations are feasible. In this appendix, we present this preliminary survey of the commercial technology currently available and how each product may or may not fit the need.

C.1.1 The Hardware

As mentioned in the report, the hardware must be capable of matching or surpassing the performance of current target display devices. This means that the graphics terminal and the host to which it is linked must provide both power and speed. The need for the ability to support almost any display design means that many colors must be available. The need for speed means that the graphic transformations and other operations must be performed in hardware. By limiting or not limiting the display designs, the choice of the graphics terminal and host will determine the success of the implemented system.

C.1.2 The Software

It is equally important that the system be easy to use. From this standpoint, the choice of the computer-aided design (CAD) package will determine the system's success. The CAD package must meet the following criteria:

1. It must be very easy to use.
 - a. The terminology must not be targeted toward a draftsman.
 - b. All information must be prompted so the user need not memorize when and what to enter.
 - c. On-line help must be available at all times.
2. It must be open ended.
 - a. It must be possible to access the data base.
 - b. It must be possible to integrate custom software.
3. It must support all display designs.
 - a. All entities must be supported.
 - b. All attributes must be supported.
 - c. It must be able to support the display hierarchy.

C.2 The Survey

A comprehensive survey was not part of the contractually prescribed effort for this project. We made a preliminary survey to determine if any systems which meet the above requirements do indeed exist. Below we present those which meet the need. We also present those which come close but do not meet the need.

This survey is based on information as of the fall of 1985. Where many of the manufacturers do not currently supply a product which fits the needs, they will likely announce equipment surpassing the need in the near future. The state-of-the art in computer technology is rapidly changing, and any report which attempts to assess the technology becomes dated before being published, even when all products have been examined. Assessment is even less possible when the survey is not exhaustive.

We have not attempted to list all of the features of the various products, but have selected those which are the most important to this system. Many have advantages which make them very attractive for other applications. However, in the interests of saving space, we will focus on only those capabilities for which we have demonstrated a need in the report. Also, many vendors produce various models of differing capacity. Where these models are equivalent except for memory size, disk storage size, or expansibility, we mention only that model which will best fit the need.

C.2.1 Graphics Terminals

Several graphics terminal products were investigated. These include the Megatek 3355, the Silicon Graphics' IRIS 2400 Turbo, Masscomp's line, Calcomp's Vistagraphic 4500XT, Spectragraphics' DesignSet 3000, Evans & Sutherland's PS 700 and PS 340, and the Tektronix 4125. Of these, only three mention high-speed filled polygons in three dimensions in the literature we received and in person-to-person consultations: Megatek, Silicon Graphics, and Spectragraphics. The 3355 provides 16 colors in double-buffered mode while the IRIS 2400 and DesignSet 3000 provide 4096 in double-buffered mode. The IRIS 2400 transforms vectors at 65,000 per second while the DesignSet 3000 transforms 1000 per second.

C.2.2 Host Computers

Many of the above graphics terminals interface with Digital's VAX line of products. The MicroVAX II would fulfill the needs of this system. Some of the above terminals provide the host computer capabilities as well. This is particularly true for the IRIS 2400 and the DesignSet 3000.

C.2.3 Other CAD Packages

Many CAD packages exist for all of the above computers and graphics terminals. We examined Applicon's BRAVO!, Autotrol's Series 5000, CALMA, FARB, MCS's Anvil 4000, Medusa, Palette, Template, Tasvir's SUPERCADS, Unicad's M/P/E, and Unigraphics. Palette, Anvil 4000, and Template will run on many computers. Palette runs on the Tektronix line as well as others while Anvil 4000 and Template run on the Megatek, Tektronix, and other lines. More products are supported by these every year. Tasvir and Unicad have products which run on the Silicon Graphic's IRIS 2400 Turbo. Unicad and Template are very flexible, allowing a programmer to redesign the interface at will. Palette sells an Independent Program Interface which allows any program to send to Palette any command which can be entered by the user, providing not only a very flexible interface but also allowing its graphics drawing capabilities to be used wherever needed. The Anvil 4000's interface is geared specifically to the draftsman, but the prompts will be modifiable in the Anvil 5000 system expected to be announced in the spring of 1986.

C.3 Conclusion

Even without an exhaustive survey, products exist which fit the needs stated in the report. Since fewer graphics terminals exist which provide all of the capabilities, the focus should be on selection of a graphics terminal before selection of a host computer and a computer-aided design package. However, the final choices must fulfill all needs, and so the availability of suitable software should be equally important.

APPENDIX D

AN ASSESSMENT OF GRADS

D.1 Introduction

As mentioned in the body of the report, the formal design should be described using a high-level programming language. For many reasons outlined in Section 3.2.2, we suggest the choice of Ada for this language. At one time another language was thought to be the best choice: the Graphics Real-time Applications Display Support (GRADS) language. However, further analysis shows that it would not be as suitable as previously supposed. We present our reasons for our claim in this appendix.

This assessment is based upon the following readings:

1. GRADS USER'S GUIDE, IR-MA-198-1, 15 April 1983, Revision 01, Intermetrics, Inc., Cambridge, MA,
2. AIDS Laboratory User's Guide, IR-MA-249, 23 September 1983, Intermetrics, Inc., Cambridge, MA, and
3. The Standard GRADS Display Interface (Preliminary), IR-258-5, January 1983, Intermetrics, Inc., Cambridge, MA.

Before presenting the advantages and disadvantages of using GRADS for the formal display description, we will first briefly describe the GRADS environment.

D.2 A Description of GRADS

GRADS is a high-level, structured programming language specifically designed for drawing graphic images in a real-time environment. It provides commands for creating fundamental shapes such as arcs, lines, circles, and squares in both two and three dimensions. It also provides the basic control structures found in any programming language: sequential flow, conditional control, and repetitive looping to facilitate the development of well-organized code. Furthermore, it provides an interface to the host computer to allow dynamic updates of the data upon which a graphic image is based. The definition of GRADS consists of:

1. the GRADS programming language,
2. the GRADS compiler, and
3. the Standard GRADS Display Interface, SGDI.

A desired graphics display is implemented by compiling the GRADS source code to create two outputs: 1) a display program for the target device's microprocessor, and 2) a subprogram which allows the data-acquisition computer to make changes in a display's parameters.

The instruction set for the program which is executed by the display subsystem is defined by SGDI. This set provides a virtual interface between the GRADS language and the actual hardware. The manufacturers of a particular piece of equipment are free to implement the drawing of entities in any manner they choose as long as they interpret the instructions according to the SGDI.

D.3 The Advantages of GRADS

The primary advantage of GRADS is that it provides a means of programming a graphics machine without the user needing to be familiar with the specific features of the target hardware. Essentially every graphics device has its own instruction set, and the programmer must relearn how to draw images whenever he moves to a new one. GRADS would eliminate this relearning. GRADS would also eliminate the need to rewrite software when a program is transferred from one display device to another. Hence the design of GRADS provides a potential for portability of techniques and software. However, it will be shown in Section D.4 that this portability is not fully realized.

Another advantage of GRADS, this time over Ada, is that an interpreter which implements the SGDI is much simpler to write than an Ada compiler. Hence, in those situations in which an Ada compiler is not available, GRADS would be preferable. However, as Ada becomes more prevalent in the industry, Ada compilers will be available for every type of computer, in particular those which are embedded in the target display devices.

D.3.1 Similar Alternatives

GRADS is not the only specification available which provides a potential for portability. The Association of Computing Machinery Special Interest Group on Computer Graphics (ACM/SIGGRAPH) has specified a standard known as Core. Also, the International Standards Organization (ISO) has created another standard: the Graphical Kernel System (GKS). While these standards and others like them are not full programming languages, they could be used in the system. However, they have been designed for all graphics applications, with the result that they are too general to be used in a real-time environment. From this we extract a second advantage of GRADS: optimization towards speed.

Hence the primary advantages of GRADS lie in potential portability and in speed. Since these effectively summarize the requirements for the display description, GRADS should be the ideal choice.

D.4 The Disadvantages of GRADS.

However, due to a lack of support by target device vendors, GRADS is effectively non-portable. To be portable, a system must be defined in such a way that it is not dependent upon the particulars of any one device and it must also be supported by a large number of computers and graphics terminals. GRADS is limited because while one vendor produces graphics terminals which support a subset of GRADS, to our knowledge no one else is planning to design other such terminals. For GRADS to become a standard, a large portion of the graphics terminal industry would need to produce supporting products.

Also, most programmers are not familiar with the GRADS language. Even though learning a new language is not difficult, it does take time. Since programmers do not stay long on a particular project at many of the facilities where the system would be used, time spent training new programmers is very significant.

Finally, Ada has been mandated by the Department of Defense for use in embedded systems; however, GRADS is not based on the Ada language. Because of this mandate, use of GRADS would require proof that it is much more suitable than Ada for this application.

D.5 Conclusion

While GRADS has its advantages, these are outweighed by the advantages of using Ada for the formal display description. The use of Ada would allow the use of commercially available compilers whereas GRADS would require the development of an interpreter for each installation. Finally, programmers are or will be more familiar with the Ada language whereas use of GRADS would require training. These factors show that while GRADS would be a reasonable language for use in this application, Ada is the best choice for use in the formal display description.

END

8-87

DTIC